Part 2: JavaScript, PHP, SQL, Advanced PHP

EE4717/IM4717 Web Application Design
JavaScript

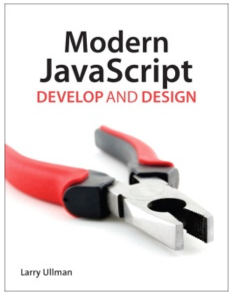
Lecturers :

Dr Karim

Dr ANG Yew Hock

References

- Recommended textbooks:



- Title: Modern JavaScript: Develop and Design

  Authors: Larry Ullman

  ISBN: 978-0321812520

  Publisher : Peachpit Press

- Title: JavaScript by Example

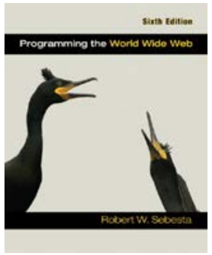
  Author: Ellie Quigley

  ISBN: 978-0-13-705489-3

  Publisher: Prentice Hall PTR

Most of the teaching slides in this part are based on materials extracted from the recommended textbooks and slides provided by authors and publishers.

- Reference Books:



- Title: Programming the World Wide Web, 6th ed.,

  Author: Robert W Sebesta

  Publisher : Addison Wesley, 2010

Overview of JavaScript

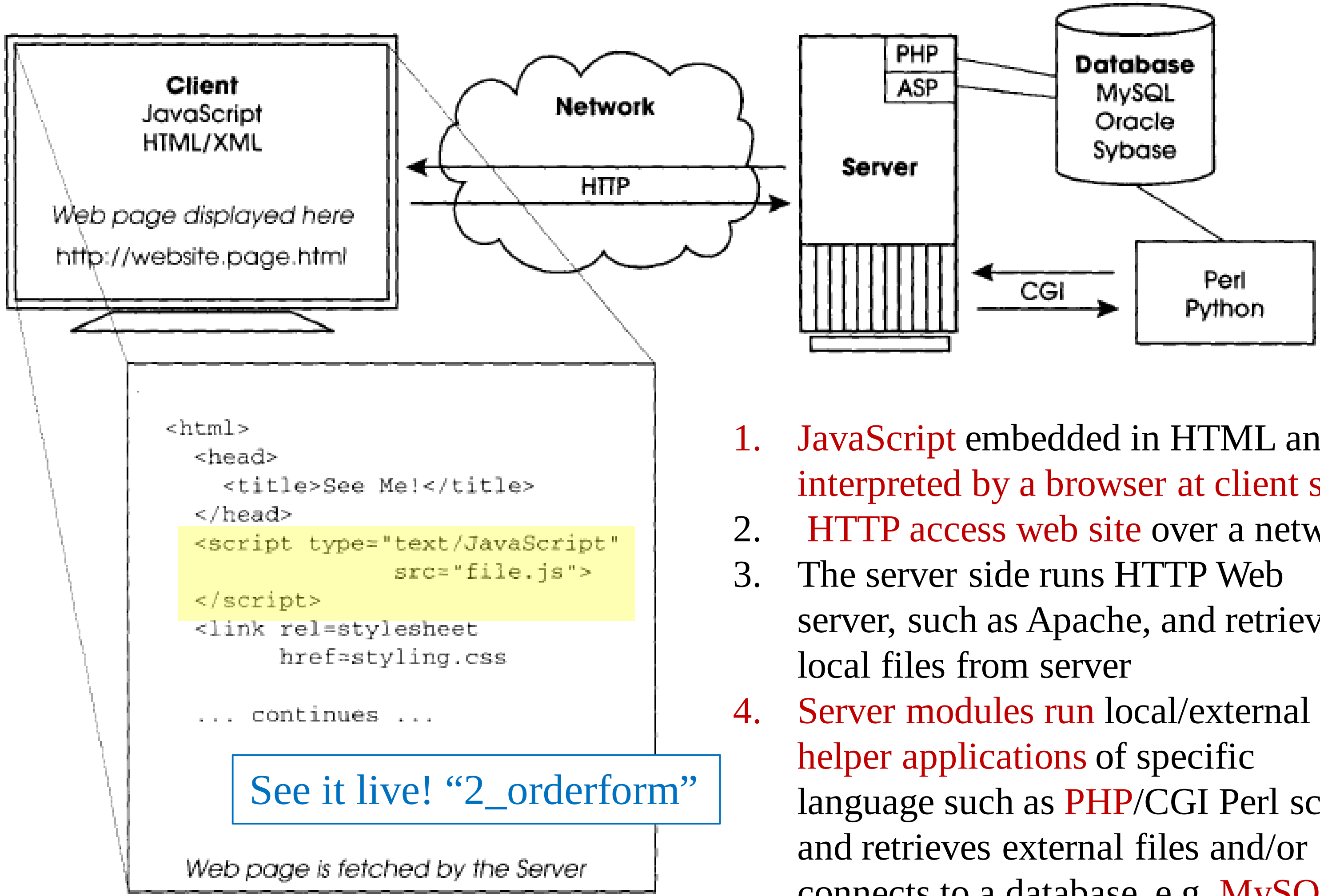
- Originally developed by Netscape, as LiveScript
  - Became a joint venture of Netscape and Sun in 1995, renamed JavaScript
- JavaScript codes are called *scripts*, not programs.
- JavaScript is object based but NOT strictly an object-oriented:
  - Does not support class-based inheritance
  - Cannot support polymorphism
- JavaScript is NOT Java.
  - JavaScript is interpreted not complied
  - JavaScript is loosely typed and flexible
- JavaScript is not HTML
  - But JavaScript code can be embedded in an HTML document within HTML tags.

JavaScript Usage

- JavaScript programs are used to detect and react to user-initiated *events*
- JavaScript lets you interact with HTML elements to control the appearance of the page as the document is being parsed.
- JavaScript lets you validate user's inputs in a form.
- JavaScript tests and directs user to plug-ins
- JavaScript has string functions to validate e-mail addresses
- JavaScript has basic constructs for variables, data types, control loops, if/else and switch statements
- JavaScript is case sensitive
- JavaScript lets your web page comes alive!

See it live! "1_scrolling text"

JavaScript in life cycle of a Web Page



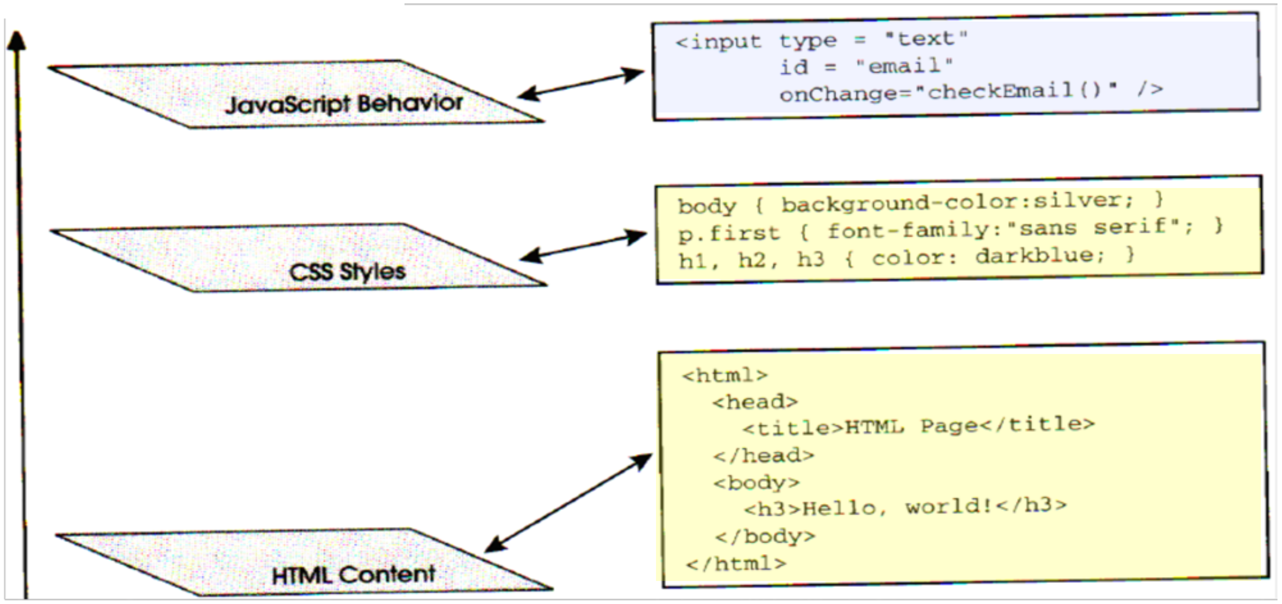
```
<html>
  <head>
    <title>See Me!</title>
  </head>
  <script type="text/JavaScript"
              src="file.js">
  </script>
  <link rel=stylesheet
        href=styling.css

  ... continues ...
```

See it live! “2_orderform”

1. JavaScript embedded in HTML and interpreted by a browser at client side
2. HTTP access web site over a network
3. The server side runs HTTP Web server, such as Apache, and retrieves local files from server
4. Server modules run local/external helper applications of specific language such as PHP/CGI Perl script, and retrieves external files and/or connects to a database, e.g. MySQL

Three Layers of a Web Page

- Three-Layer structure
  - Content, style, and behavior layers
    - Valid markup is important to content structural
    - Styles sheets allow cascading changes to presentational content
    - JS allows user interaction of behavioral content
  - Separate layers provide design and programming independence



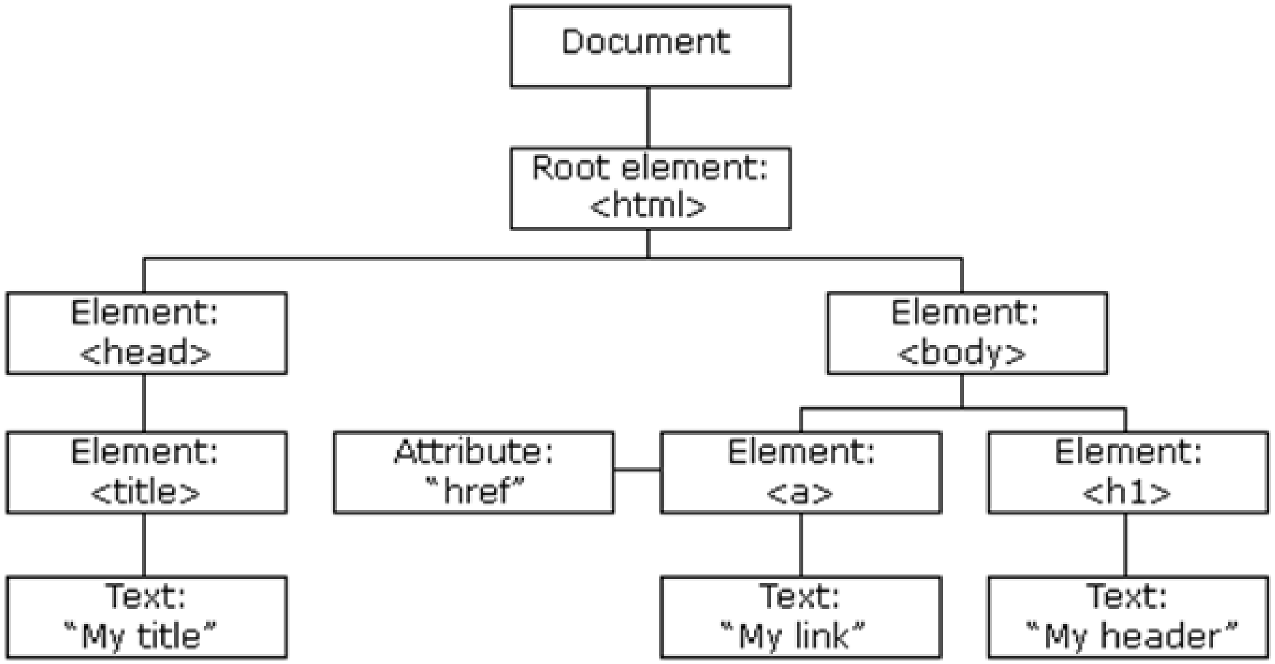
```
<input type = "text"
       id = "email"
       onChange="checkEmail()" />
```

```
body { background-color:silver; }
p.first { font-family:"sans serif"; }
h1, h2, h3 { color: darkblue; }
```

```
<html>
  <head>
    <title>HTML Page</title>
  </head>
  <body>
    <h3>Hello, world!</h3>
  </body>
</html>
```

The Document Object Model (DOM)

- The HTML DOM model is constructed as a tree of Objects and allows style of a document to be dynamically changes.
  - All nodes in the tree can be accessed by JavaScript



Object and JavaScript

- The root object in JavaScript is **`Object`** - objects are derived from **`Object`**
  - An object has property and method
  - all HTML elements in DOM are objects
- JavaScript scripts are embedded in HTML documents
  - Either directly, as in

```
<script type = "text/javaScript">
     -- JavaScript script –
</script>
```

  - Or indirectly, as a file specified in the **`src`** attribute of **`<script>`**, as in

```
<script type="text/javaScript” src=“myScript.js”>
```

Comments in JavaScript

- A single line comment starts with //
  - Any text between // and the end of a line, will be ignored by JavaScript (will not be executed).

```
// This is a comment
```

- A block of comments is enclosed by /* */ symbols

```
/* This is a block of comments
   that continues for a number of lines
*/
```

- Hiding JS from browsers without scripts support

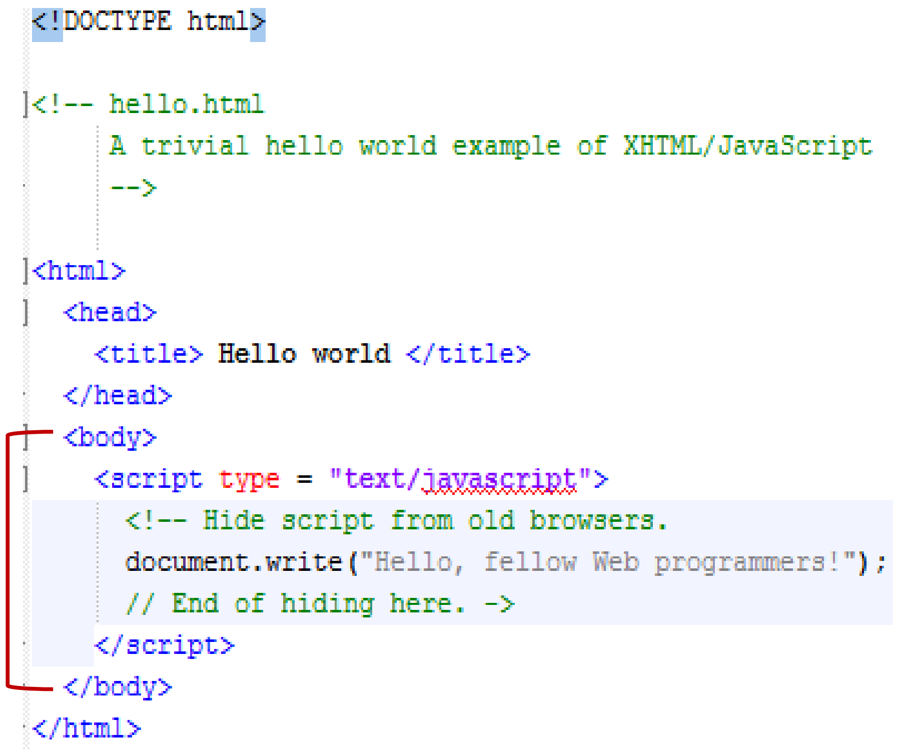
```
<script
   <!-- Hide script from old browsers.
    Document.write("Hello World!");
   // End of hiding here. -->
</script>
```

Example 1.1 - JavaScript and HTML

- JavaScript is embedded in HTML
  - Between head tags, <head> and </head>
  - within the body tags, <body> and </body>
  - Or, in an external text file (text file with *.js*)
- “Hello World”

Try it! “Ex1.1_hello”

- “Hello World”

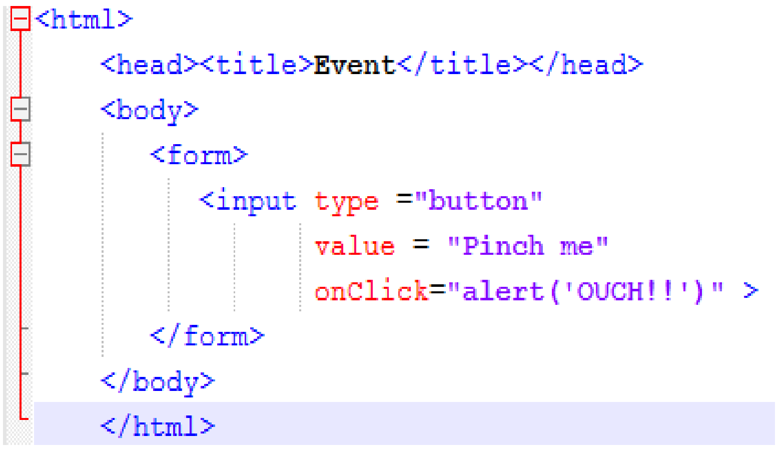


```
<!DOCTYPE html>

<!-- hello.html
     A trivial hello world example of XHTML/JavaScript
     -->

<html>
  <head>
    <title> Hello world </title>
  </head>
  <body>
    <script type = "text/javascript">
      <!-- Hide script from old browsers.
      document.write("Hello, fellow Web programmers!");
      // End of hiding here. ->
    </script>
  </body>
</html>
```

Example 1.2 - JavaScript and Events

- JavaScript responds to events and interact dynamically with the user.
- E.g., JavaScript *onClick* event handler handles attributes of HTML *<form>* tag of type "*button*".
  - On a click of the button, JavaScript event, called *click*, a value is assigned to *onClick* event handler to act.

- "Pinch Me"



```
<html>
    <head><title>Event</title></head>
    <body>
        <form>
            <input type ="button"
                    value = "Pinch me"
                    onClick="alert('OUCH!!')" >
        </form>
    </body>
    </html>
```

Try it! "Ex1.2_pinch me"

JavaScript Event Handlers(1)

Some common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

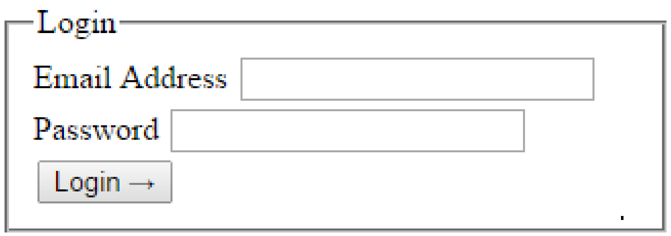
JavaScript Event Handlers(2)

Some common Form events

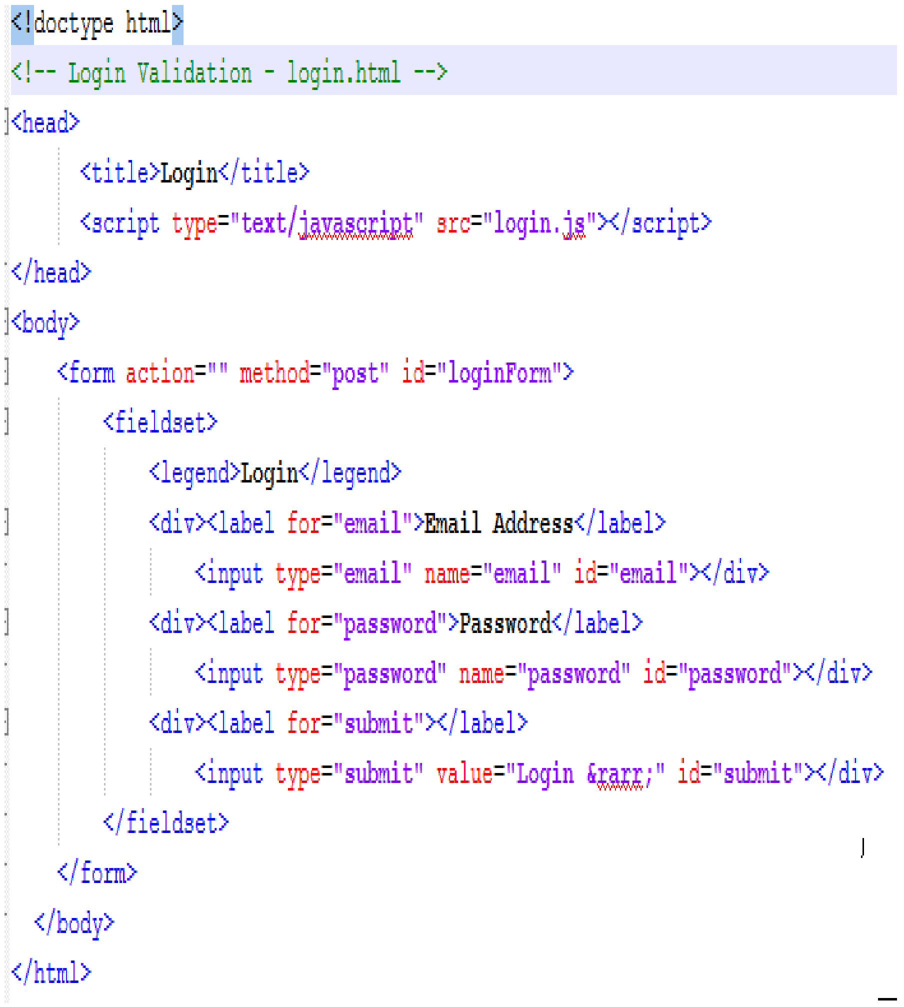
Event Handler	What caused it
onblur	The event occurs when a form element loses focus
onchange	The event occurs when an element in the form changed value
onfocus	The event occurs when an element gets focus
onfocusin	The event occurs when an element is about to get focus
onfocusout	The event occurs when an element is about to lose focus
oninput	The event occurs when an element gets user input
onreset	The event occurs when a form is reset
onsubmit	The event occurs when a form is submitted

Event Analysis – Form validation(1)

- Example of a simple login form that takes in an email address and a password, and has a Login button
- On submission, inputs are validated using an external JavaScript file, *login.js*



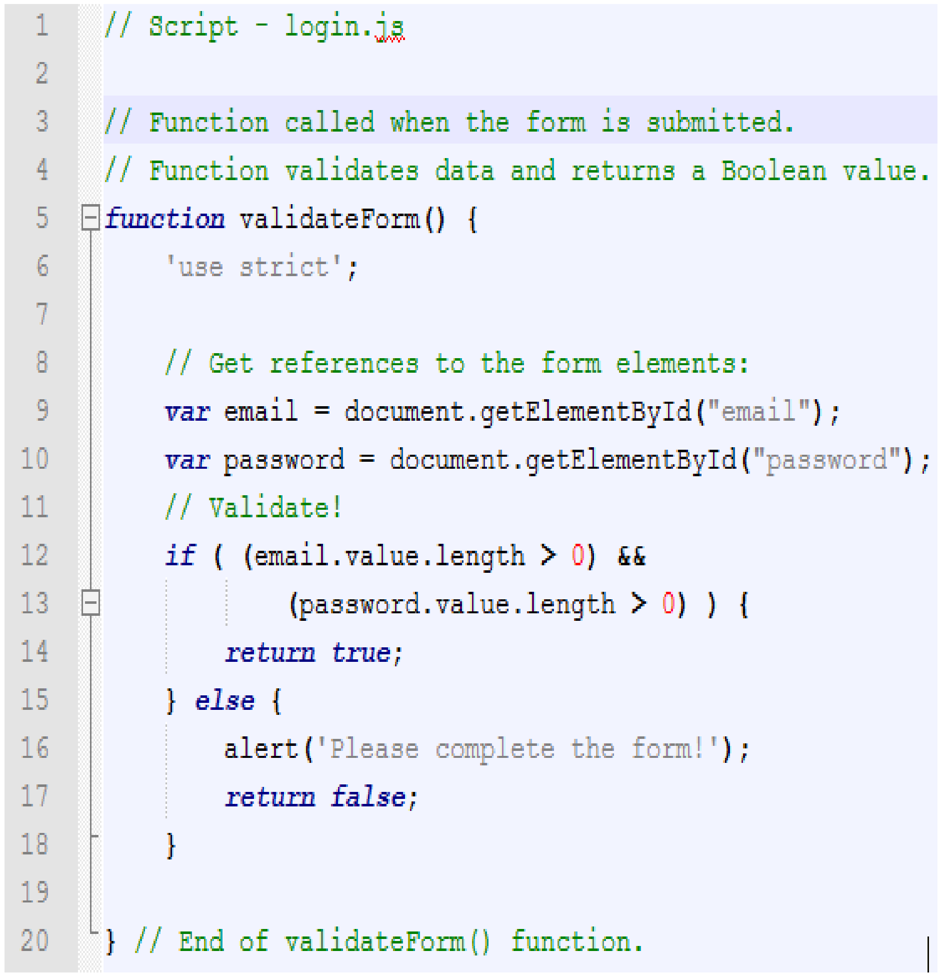
See it live! “3_login”



```
<!doctype html>
<!-- Login Validation - login.html -->
<head>
    <title>Login</title>
    <script type="text/javascript" src="login.js"></script>
</head>
<body>
  <form action="" method="post" id="loginForm">
    <fieldset>
      <legend>Login</legend>
      <div><label for="email">Email Address</label>
        <input type="email" name="email" id="email"></div>
      <div><label for="password">Password</label>
        <input type="password" name="password" id="password"></div>
      <div><label for="submit"></label>
        <input type="submit" value="Login &rarr;" id="submit"></div>
    </fieldset>
  </form>
 </body>
</html>
```

Event Analysis – Form validation(2)

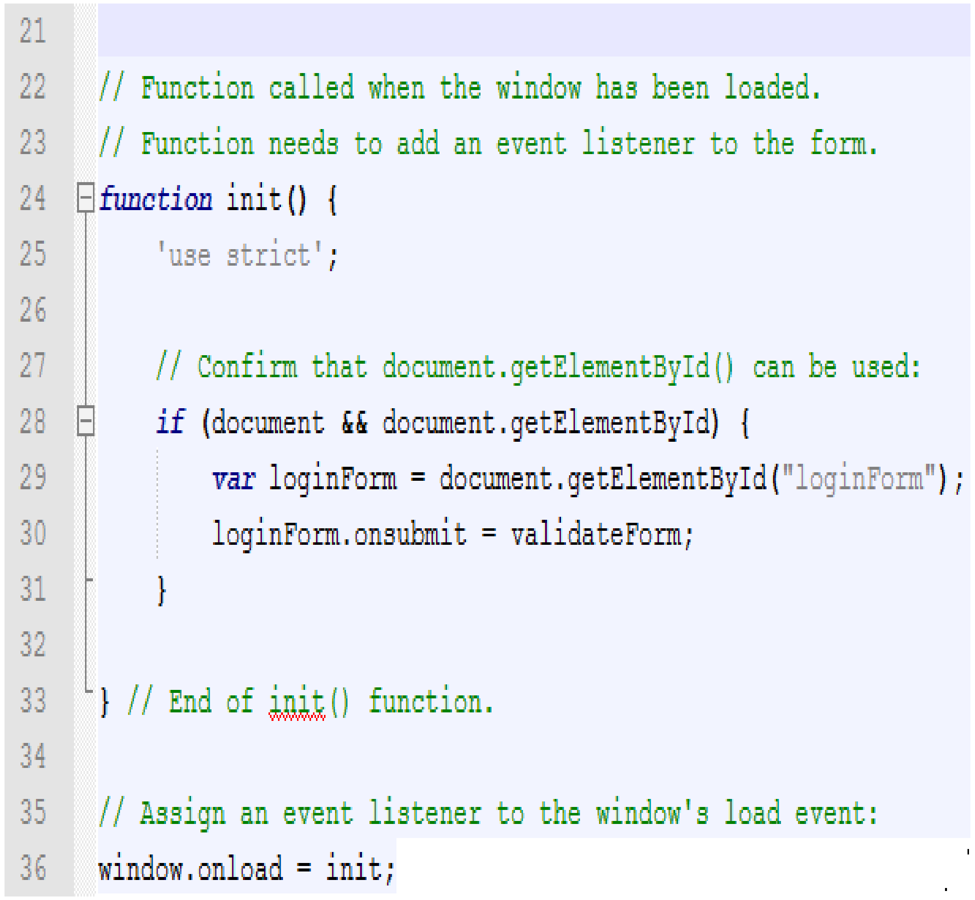
- Form inputs are grabbed from document object using *getElemenById()* method.
- Input variables can be referenced by their element's names, such as "email" and "password"
- Validation to ensure input strings are not empty by referring to the property of the element value, *email.value* and *password.value*
- The length of the input strings are checked by referring to the value in the *length* property



```
// Script - login.js

// Function called when the form is submitted.
// Function validates data and returns a Boolean value.
function validateForm() {
    'use strict';

    // Get references to the form elements:
    var email = document.getElementById("email");
    var password = document.getElementById("password");
    // Validate!
    if ( (email.value.length > 0) &&
            (password.value.length > 0) ) {
        return true;
    } else {
        alert('Please complete the form!');
        return false;
    }

} // End of validateForm() function.
```

Event Analysis – Form validation(3)

- Event listener set to watch for browser to complete loading the entire page
- On occurrence of *window.onload* event trigger *init()* function
- *Init()* function gets document object element *loginForm* using the *getElemenById()* method
- It adds an event listener for *loginForm.onsubmit* to wait for submission of the form
- On submission of form, call *validationForm()* function



```
// Function called when the window has been loaded.
// Function needs to add an event listener to the form.
function init() {
    'use strict';

    // Confirm that document.getElementById() can be used:
    if (document && document.getElementById) {
        var loginForm = document.getElementById("loginForm");
        loginForm.onsubmit = validateForm;
    }

} // End of init() function.

// Assign an event listener to the window's load event:
window.onload = init;
```

JavaScript @ w3schools

- All about JavaScript @ w3schools.com
  - JS Tutorial with hundred of "Try it yourself" examples
  - HTML DOM
  - Objects
  - Window
  - Libraries
  - Examples
  - References
  - Quiz
  - Tryit! Editor

http://www.w3schools.com/js/default.asp

Primitives, Operations, Expressions

- Primitive data types
  - Numeric, string, boolean, including null and undefined
  - Primitive variables only store a single literal value
- JavaScript is dynamically typed
  - Any variable can be used for anything (primitive value or reference to any object)
  - The interpreter determines the type of a particular occurrence of a variable
- Variables can be either implicitly or explicitly declared

```
var sum = 0,
today = "Monday";
flag = false;
```

Primitives, Operations, Expressions

- Numeric operators : `++`, `--`, `+`, `-`, `*`, `/`, `%`
  - All operations are in double precision
  - Has order of precedence (use parentheses if not sure)
- `Math` Object provides `floor`, `round`, `max`, `min`, trig functions, etc.

  e.g., `Math.cos(x)`
- The `Number` Object
  - Some useful properties:

  `MAX_VALUE`, `MIN_VALUE`, `NaN`,

  `POSITIVE_INFINITY`, `NEGATIVE_INFINITY`, `PI`

  e.g., `Number.MAX_VALUE`

Primitives, Operations, Expressions

- String concatenation operator is “`+`”
- Coercions
  - Concatenation coerces numbers to strings (e.g. 123 to “123”)
  - Numeric operators (other than `+`) coerce strings to numbers (if either operand of `+` is a string, it is assumed to be concatenation)
  - Conversions from strings to numbers that do not work return `NaN`
- Explicit conversions
  1. Use the `String` and `Number` constructors
  2. Use `toString` method of numbers
  3. Use `parseInt` and `parseFloat` on strings

Primitives, Operations, Expressions

- String properties & methods:
  - `length` e.g., `var len = str.length;` (a property, has value)
  - `charAt(`position`)` e.g., `str.charAt(3)` (a method, has function)
  - `indexOf(`string`)` e.g., `str.indexOf('B')`
  - `substring(`from, to`)` e.g., `str.substring(1, 3)`
  - `toLowerCase()` e.g., `str.toLowerCase()`
- The `typeof` operator
  - Returns `"number"`, `"string"`, or `"boolean"` for Number, String, or Boolean, `"undefined"` for Undefined, `"function"` for functions, and `"object"` for objects and NULL

```
var x = 123;
var y = new Number(123); (a new Number object)
typeof(x) // returns Number
typeof(y) // returns Object
```

Screen Output & Keyboard Input

- The JavaScript model for the HTML document is the **`Document`** object
- The model for the browser display window is the **`Window`** object
  - The **`Window`** object has two properties, **`document`** and **`window`**, which refer to the **`Document`** and **`Window`** objects, respectively
- The **`Document`** object has a method, **`write`**, which dynamically creates content
- The parameter is a string, often concatenated from parts, some of which are variables

  e.g., `document.write("Answer: " + result + "<br />");`
- The parameter is sent to the browser, so it can be anything that can appear in an HTML document (including HTML tags)
  - Note: `<br />` is written as `<br>` since HTML 5

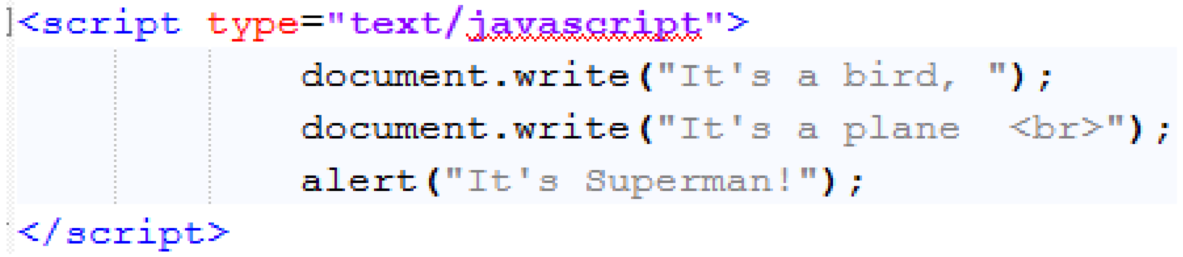
Screen Output & Keyboard Input

The **`Window`** object has three methods for creating dialog boxes, **`alert`**, **`confirm`**, and **`prompt`**

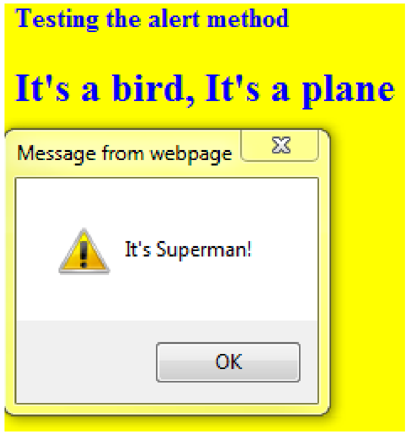
- 1. **`alert`**`("String of pain text");`

  `alert``(expression);`

  - Parameter is plain text, not HTML
  - Opens a dialog box which displays the parameter string and an `OK` button
  - It waits for the user to press the `OK` button
  - Example (document outputs and window alert)



```
<script type="text/javascript">
            document.write("It's a bird, ");
            document.write("It's a plane <br>");
            alert("It's Superman!");
</script>
```



Testing the alert method

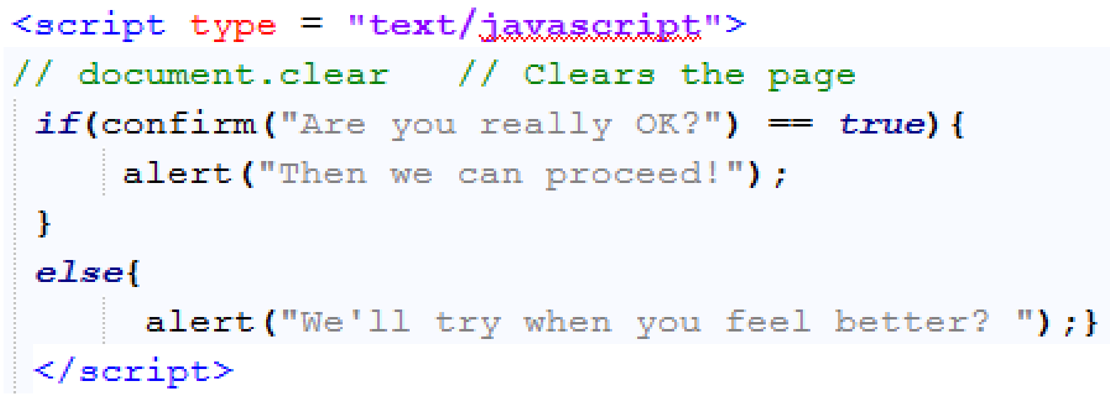
It's a bird, It's a plane

Message from webpage

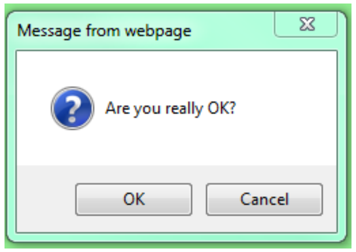
It's Superman!

Screen Output & Keyboard Input

- 2. **`confirm("Do you want to continue?");`**
  - Opens a dialog box and displays the parameter and two buttons, **`OK`** and **`Cancel`**
  - Returns a Boolean value, depending on which button was pressed (it waits for one)
  - Example

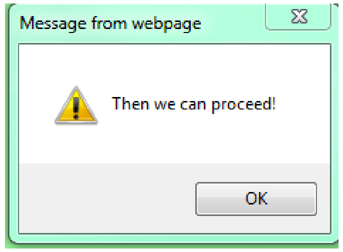


```
<script type = "text/javascript">
// document.clear    // Clears the page
 if(confirm("Are you really OK?") == true){
     alert("Then we can proceed!");
 }
 else{
      alert("We'll try when you feel better? ");}
 </script>
```



Message from webpage

Are you really OK?



Message from webpage

Then we can proceed!

See it live! “4_confirm”

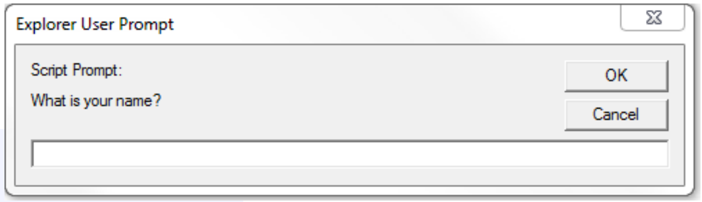
Screen Output & Keyboard Input

- 3. **prompt**("message", "");

  prompt("message", "defaultText");

  - Opens a dialog box and displays its string parameter, along with a **text box** and two buttons, **OK** and **Cancel**
  - The second parameter is for a default response if the user presses `OK` without typing a response in the text box (waits for `OK`)
  - Example

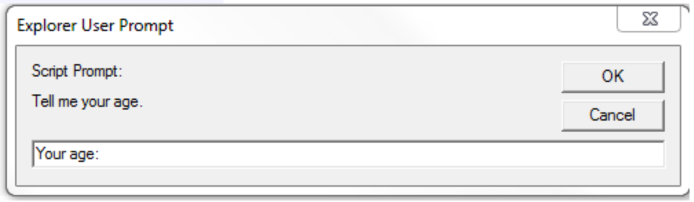
```
<script type="text/javascript">
var name=prompt("What is your name?", "");
alert("Welcome to my world! " + name);
var age=prompt("Tell me your age.", "Your age: ");
if ( age == null){     // If user presses the cancel button
    alert("Not sharing your age with me");
}
else{
    alert(age + " is young");
}
alert(prompt("Where do you live? ", ""));
</script>
```



Explorer User Prompt

Script Prompt:

What is your name?



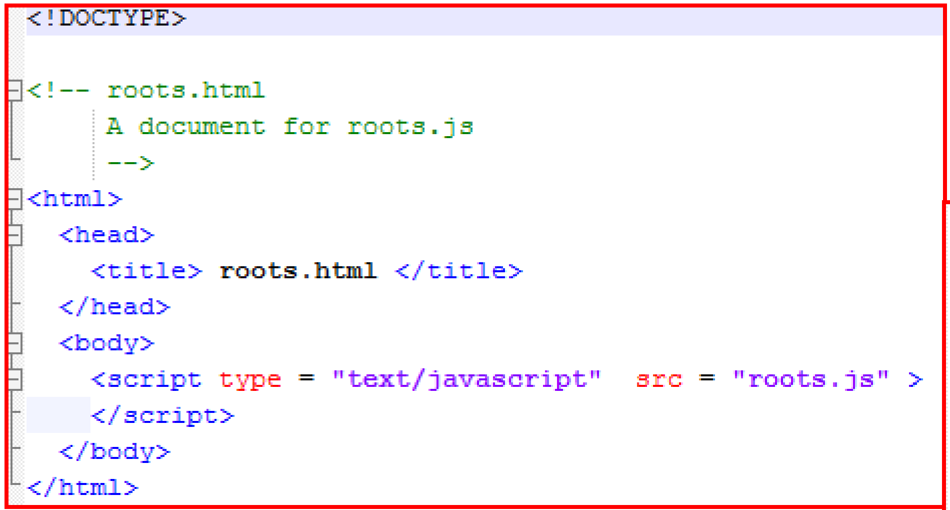
Explorer User Prompt

Script Prompt:

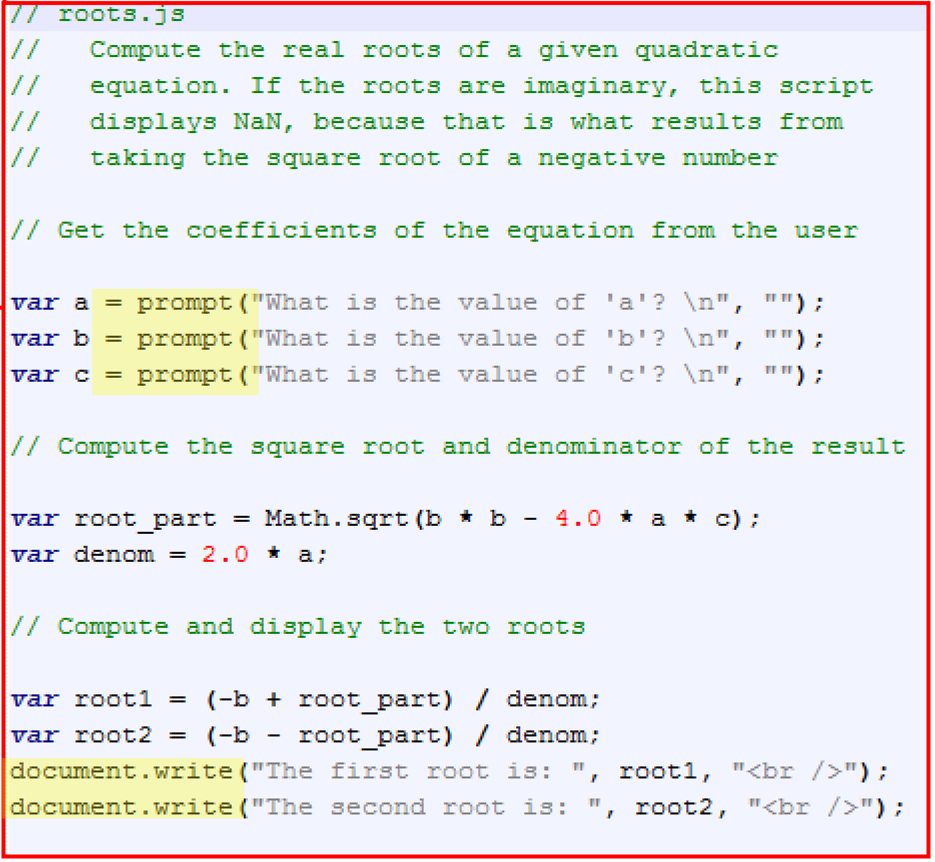
Tell me your age.

Your age:

Example 2 – Input & Output



```
<!DOCTYPE>

<!-- roots.html
     A document for roots.js
     -->
<html>
  <head>
    <title> roots.html </title>
  </head>
  <body>
    <script type = "text/javascript"  src = "roots.js" >
    </script>
  </body>
</html>
```



```
// roots.js
//   Compute the real roots of a given quadratic
//   equation. If the roots are imaginary, this script
//   displays NaN, because that is what results from
//   taking the square root of a negative number

// Get the coefficients of the equation from the user

var a = prompt("What is the value of 'a'? \n", "");
var b = prompt("What is the value of 'b'? \n", "");
var c = prompt("What is the value of 'c'? \n", "");

// Compute the square root and denominator of the result

var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;

// Compute and display the two roots

var root1 = (-b + root_part) / denom;
var root2 = (-b - root_part) / denom;
document.write("The first root is: ", root1, "<br />");
document.write("The second root is: ", root2, "<br />");
```

Try it! “Ex2_roots”

Control Statements(1)

- Three types of *Control expressions*
  1. Primitive values
     - If it is a string, it is true unless it is empty or `"0"`
     - If it is a number, it is true unless it is zero
  2. Relational Expressions
     - The usual six: `==`, `!=`, `<`, `>`, `<=`, `>=`
     - Operands are coerced if necessary
       - a string or a boolean is converted to a number when operate with a number
  3. Compound Expressions
     - The usual operators: `&&`, `||`, and `!`
     - order of operation has significance

Control Statements(2)

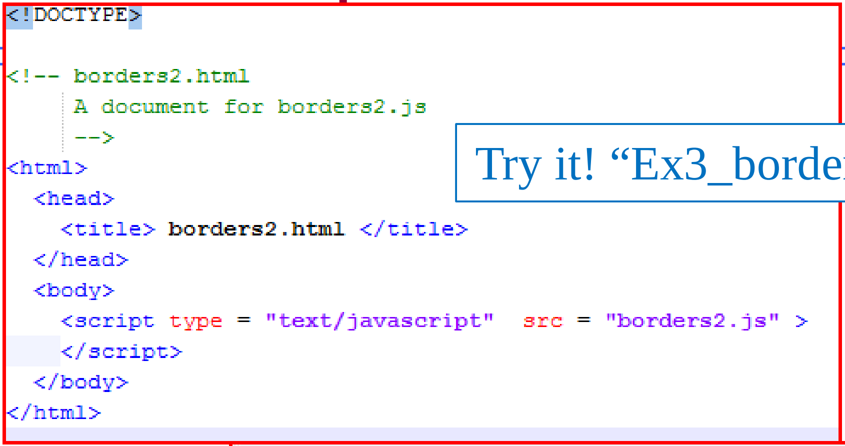
- Switch

```
switch (expression) {
   case value_1:
      // value_1 statements
   case value_2:
      // value_2 statements
   …
   [default:
      // default statements]
}
```

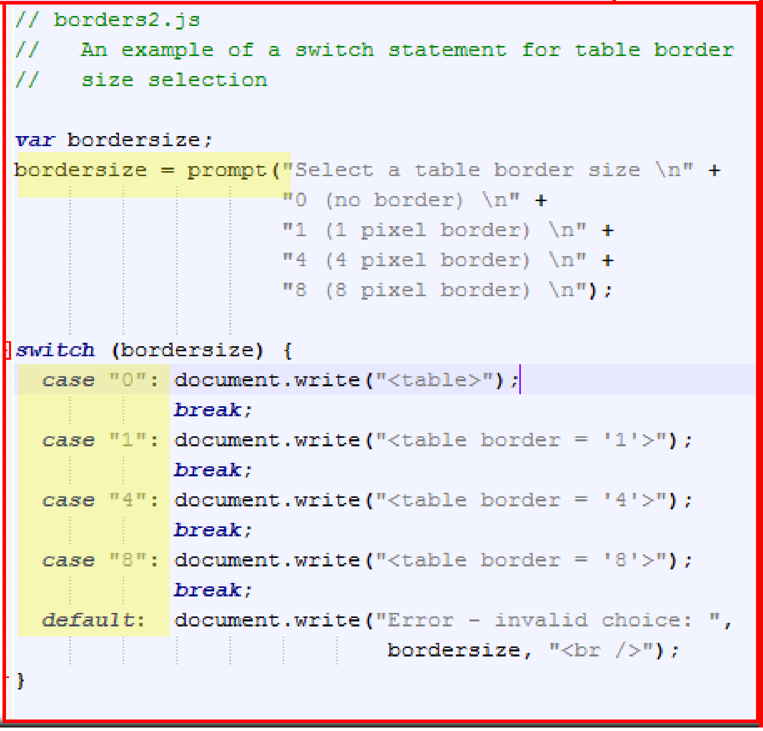
  - The statements can be either statement sequences or compound statements
  - The control expression can be a number, a string, or a Boolean
  - Different cases can have values of different types

Example 3 – *switch* statement

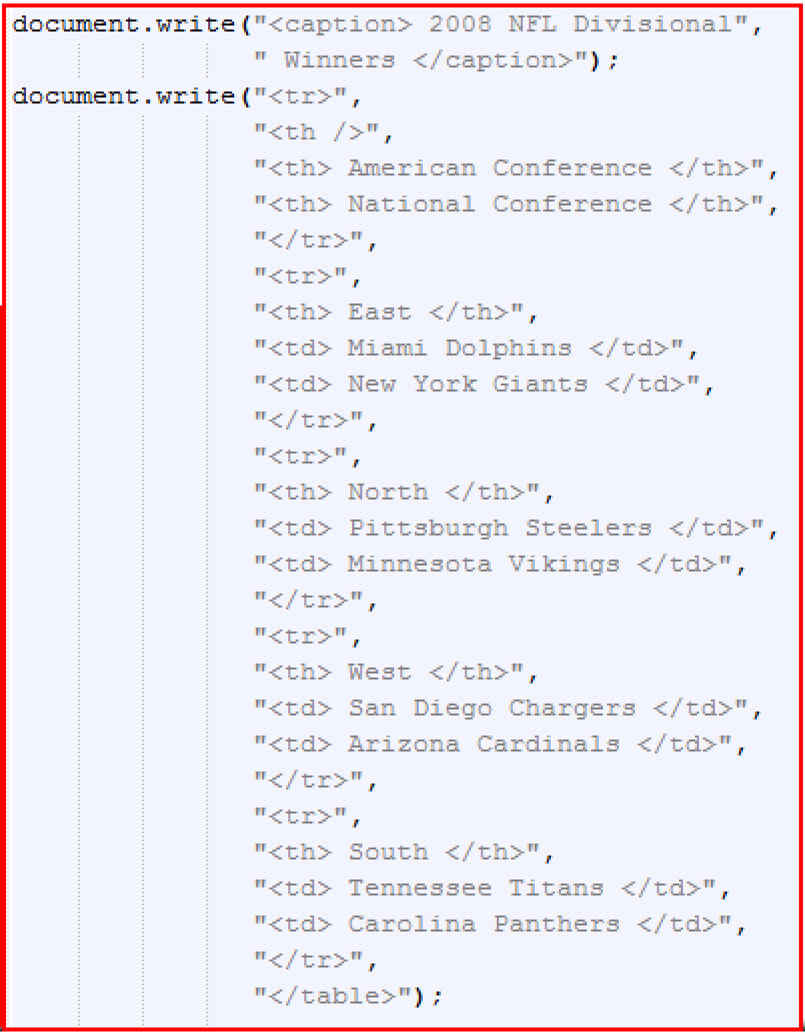
Try it! “Ex3_borders”



```
<!DOCTYPE>

<!-- borders2.html
     A document for borders2.js
     -->
<html>
  <head>
    <title> borders2.html </title>
  </head>
  <body>
    <script type = "text/javascript"  src = "borders2.js" >
    </script>
  </body>
</html>
```



```
// borders2.js
//    An example of a switch statement for table border
//    size selection

var bordersize;
bordersize = prompt("Select a table border size \n" +
                    "0 (no border) \n" +
                    "1 (1 pixel border) \n" +
                    "4 (4 pixel border) \n" +
                    "8 (8 pixel border) \n");

switch (bordersize) {
  case "0": document.write("<table>");
            break;
  case "1": document.write("<table border = '1'>");
            break;
  case "4": document.write("<table border = '4'>");
            break;
  case "8": document.write("<table border = '8'>");
            break;
  default:  document.write("Error - invalid choice: ",
                           bordersize, "<br />");
}
```



```
document.write("<caption> 2008 NFL Divisional",
               " Winners </caption>");
document.write("<tr>",
               "<th />",
               "<th> American Conference </th>",
               "<th> National Conference </th>",
               "</tr>",
               "<tr>",
               "<th> East </th>",
               "<td> Miami Dolphins </td>",
               "<td> New York Giants </td>",
               "</tr>",
               "<tr>",
               "<th> North </th>",
               "<td> Pittsburgh Steelers </td>",
               "<td> Minnesota Vikings </td>",
               "</tr>",
               "<tr>",
               "<th> West </th>",
               "<td> San Diego Chargers </td>",
               "<td> Arizona Cardinals </td>",
               "</tr>",
               "<tr>",
               "<th> South </th>",
               "<td> Tennessee Titans </td>",
               "<td> Carolina Panthers </td>",
               "</tr>",
               "</table>");
```

Control Statements(3)

- Loop statements

```
while (control_expression) {
        statement or compound;
        increment/decrement counter;
 }

for (init; control; increment/decrement) {
        statement or compound;
 }

do {
    statement or compound;
}
while (control_expression)
```

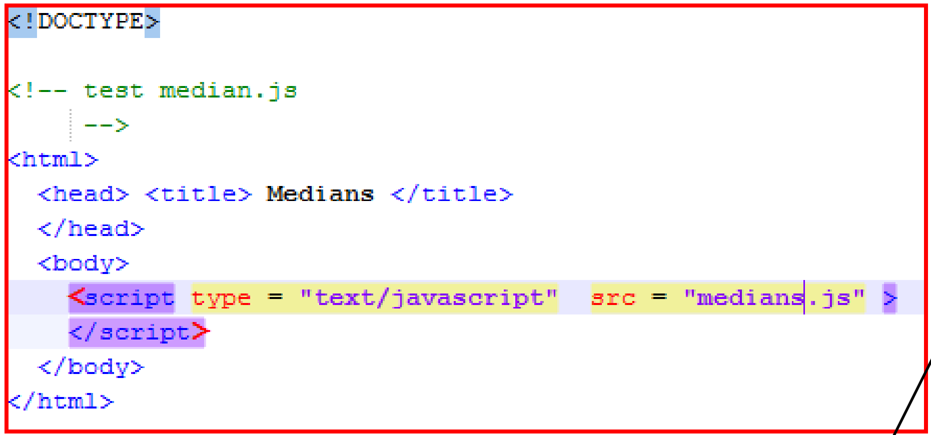
Functions

- A function is a block of code that execute when "someone" calls it
  - block code (inside curly { } braces), preceded by the function keyword
  - Must be declared before they are used and normally in the <head> tag
- Format - *function name begins with a **small** letter*

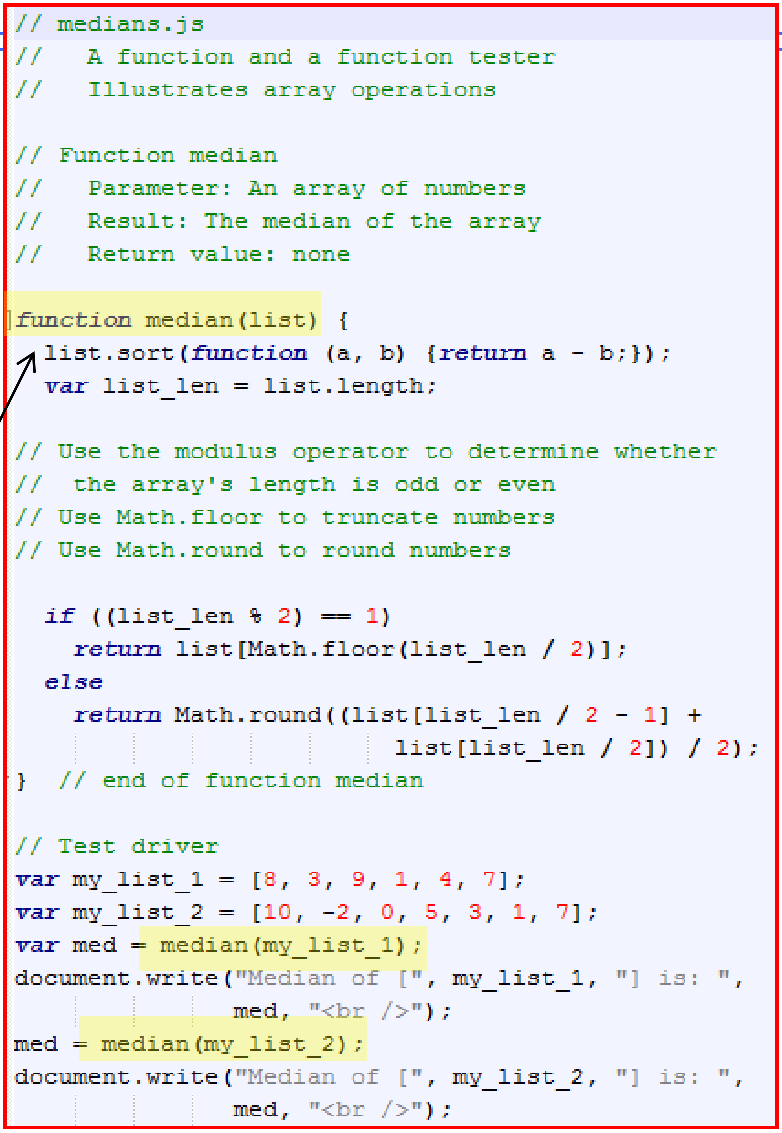
```
function function_name([formal_parameters]) {
        statements; statements;
}
```

- Parameters can be passed by value or pass-by-reference variables.
- No type checking nor number of parameters checked (excess actual parameters are ignored, excess formal parameters are set to `undefined`).
- All parameters are sent through a property array, `arguments`, which has the `length` property.
  - Return value is the parameter of `return`
  - If there is no `return` at the end of function, `undefined` is returned
  - If `return` has no parameter, `undefined` is returned

Example 4 – *Functions*



```
<!DOCTYPE>

<!-- test median.js
    -->
<html>
  <head> <title> Medians </title>
  </head>
  <body>
    <script type = "text/javascript"  src = "medians.js" >
    </script>
  </body>
</html>
```



```
// medians.js
//   A function and a function tester
//   Illustrates array operations

// Function median
//   Parameter: An array of numbers
//   Result: The median of the array
//   Return value: none

function median(list) {
  list.sort(function (a, b) {return a - b;});
  var list_len = list.length;

// Use the modulus operator to determine whether
//  the array's length is odd or even
// Use Math.floor to truncate numbers
// Use Math.round to round numbers

  if ((list_len % 2) == 1)
    return list[Math.floor(list_len / 2)];
  else
    return Math.round((list[list_len / 2 - 1] +
                          list[list_len / 2]) / 2);
} // end of function median

// Test driver
var my_list_1 = [8, 3, 9, 1, 4, 7];
var my_list_2 = [10, -2, 0, 5, 3, 1, 7];
var med = median(my_list_1);
document.write("Median of [", my_list_1, "] is: ",
               med, "<br />");
med = median(my_list_2);
document.write("Median of [", my_list_2, "] is: ",
               med, "<br />");
```

Note: use of optional *compareFunction*

`list.sort`(*compare-function option*);

Try it! “Ex4_medians”

Functions as Constructors

- Used to initialize objects, but actually create the properties

Define:
```
function plane(newMake, newModel, newYear){
      this.make = newMake;
      this.model = newModel;
      this.year = newYear;
}
```

"*this*" is a current object

Use:
```
myPlane = new plane("Cessna", "Centurian", "1970");
```

- Function as object methods

Define:
```
function displayPlane() {
      document.write("Make: ", this.make, "<br />");
      document.write("Model: ", this.model, "<br />");
      document.write("Year: ", this.year, "<br />");
}
```

Parentheses differentiate a method from a property

  – Now add the method to the object by assigning to the constructor :

```
myPlane.display = displayPlane;
```

See it live! "4_fnConstructor"

Object Creation and Modification

- Objects can be created with the `new` operator followed by a function
- The most basic object is one that uses the `Object()` constructor, as in

```
var myObject = new Object();
```

- The new object has no properties - a blank object
- Properties can be added to an object, any time

```
var myAirplane = new Object();
    myAirplane.make = "Cessna";
    myAirplane.model = "Centurian";
```

- Properties can be accessed by dot or in array notations:

```
var property1 = myAirplane["model"];
delete myAirplane.model;
```

JavaScript Objects

- Objects are complex data types
- JavaScript core objects are built-in JavaScript objects (names begin with a capital letter), such as
  - `Date()`
  - `Array()`
  - `Function()`
  - `RegExp()`
- Even primitive data types (except null and undefined) can be treated as objects
  - Booleans
  - Numbers
  - Strings
- JavaScript allows you to define your own objects

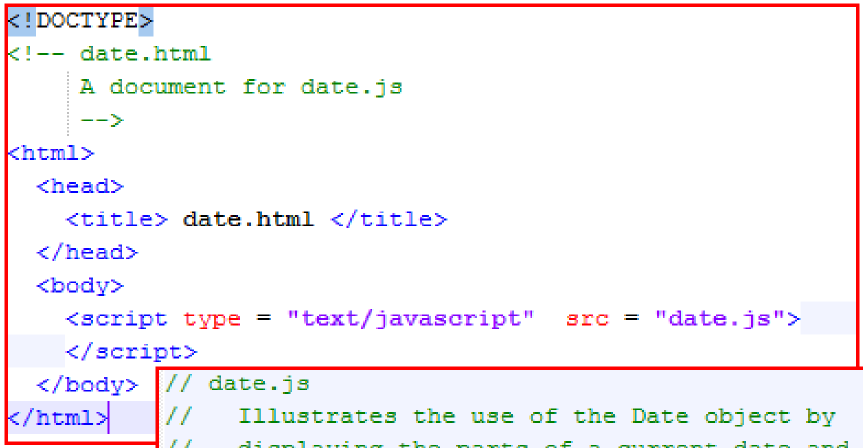
JS Core Object – *Date* Object

- The `Date` Object returns times and dates local to the browser
  - Create one with the `Date` constructor (no params)
  - Format

```
var now = new Date();
```

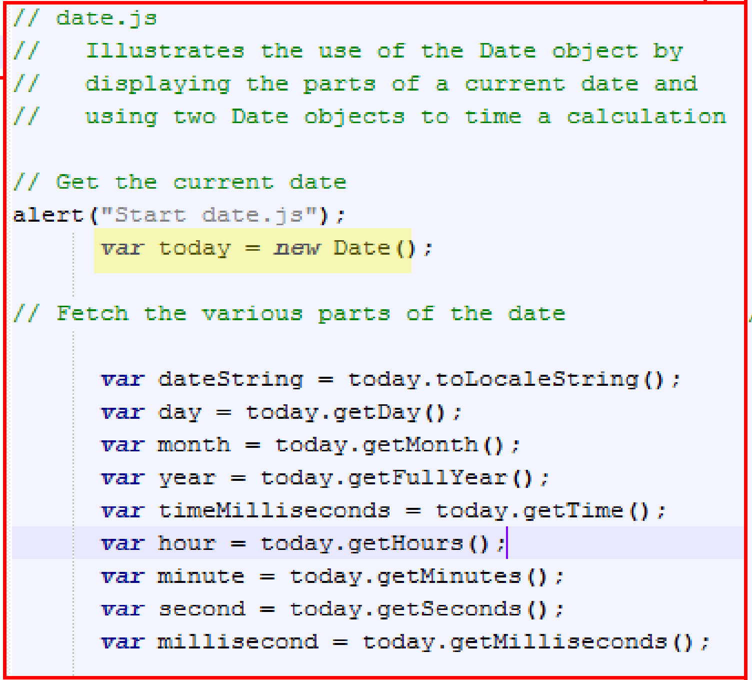
  - Local time methods of `Date`:
    - toLocaleString – returns a string of the date
    - getDate – returns the day of the month
    - getMonth – returns the month of the year (0 – 11)
    - getDay – returns the day of the week (0 – 6)
    - getFullYear – returns the year
    - getTime – returns the number of millisecond since Jan 1, 1970
    - getHours – returns the hour (0 – 23)
    - getMinutes – returns the minutes (0 – 59)
    - getMilliseconds – returns the millisecond (0 – 999)

Example 5 – *Date* Object

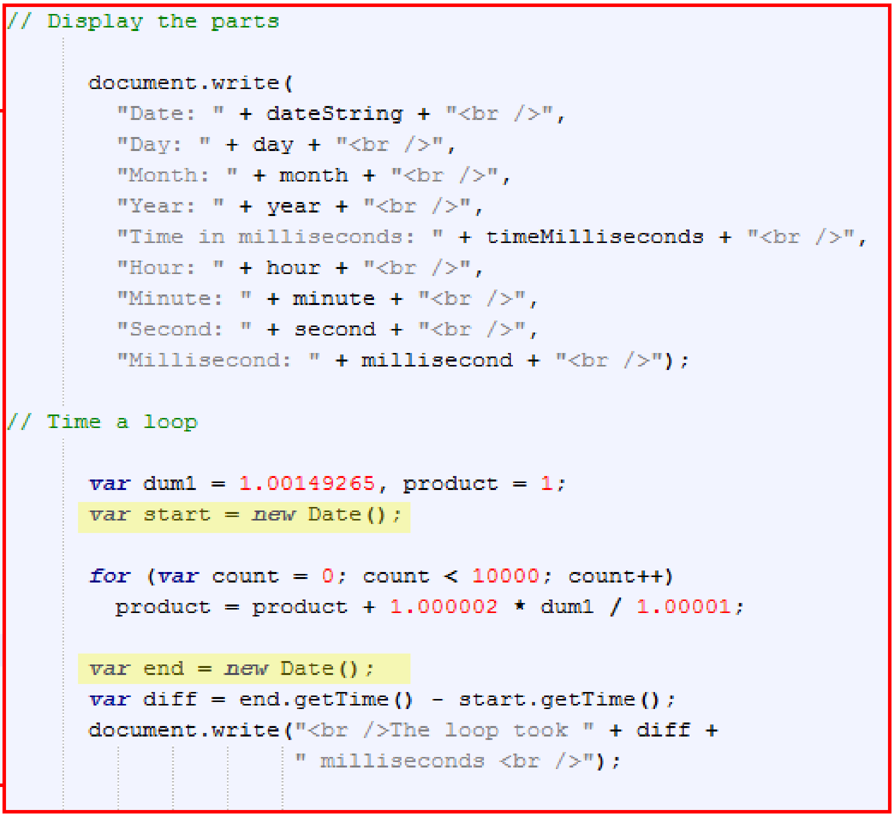


```
<!DOCTYPE>
<!-- date.html
     A document for date.js
     -->
<html>
  <head>
    <title> date.html </title>
  </head>
  <body>
    <script type = "text/javascript"  src = "date.js">
    </script>
  </body>
</html>
```

Try it! “Ex5_date”



```
// date.js
//   Illustrates the use of the Date object by
//   displaying the parts of a current date and
//   using two Date objects to time a calculation

// Get the current date
alert("Start date.js");
      var today = new Date();

// Fetch the various parts of the date

      var dateString = today.toLocaleString();
      var day = today.getDay();
      var month = today.getMonth();
      var year = today.getFullYear();
      var timeMilliseconds = today.getTime();
      var hour = today.getHours();
      var minute = today.getMinutes();
      var second = today.getSeconds();
      var millisecond = today.getMilliseconds();
```



```
// Display the parts

      document.write(
        "Date: " + dateString + "<br />",
        "Day: " + day + "<br />",
        "Month: " + month + "<br />",
        "Year: " + year + "<br />",
        "Time in milliseconds: " + timeMilliseconds + "<br />",
        "Hour: " + hour + "<br />",
        "Minute: " + minute + "<br />",
        "Second: " + second + "<br />",
        "Millisecond: " + millisecond + "<br />");

// Time a loop

      var dum1 = 1.00149265, product = 1;
      var start = new Date();

      for (var count = 0; count < 10000; count++)
        product = product + 1.000002 * dum1 / 1.00001;

      var end = new Date();
      var diff = end.getTime() - start.getTime();
      document.write("<br />The loop took " + diff +
                     " milliseconds <br />");
```

JS Core Object – *Array* Object

- Array elements can be primitive values or references to other objects
- Length is dynamic
  - the `length` property stores the length
- Array objects can be created in two ways, with `new`, or by assigning an array literal

```
var myList = new Array(24, "bread", true);
var myList2 = [24, "bread", true];
```

- The length of an array is the highest index (starts at 0), plus 1

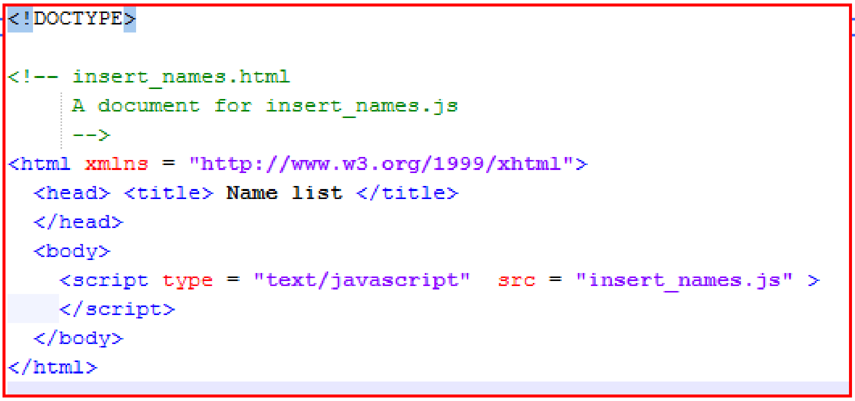
```
myList[122] = "bitsy";  // length is 123
```

- Because the `length` property is writeable, you can set it to make the array any length you like, as in

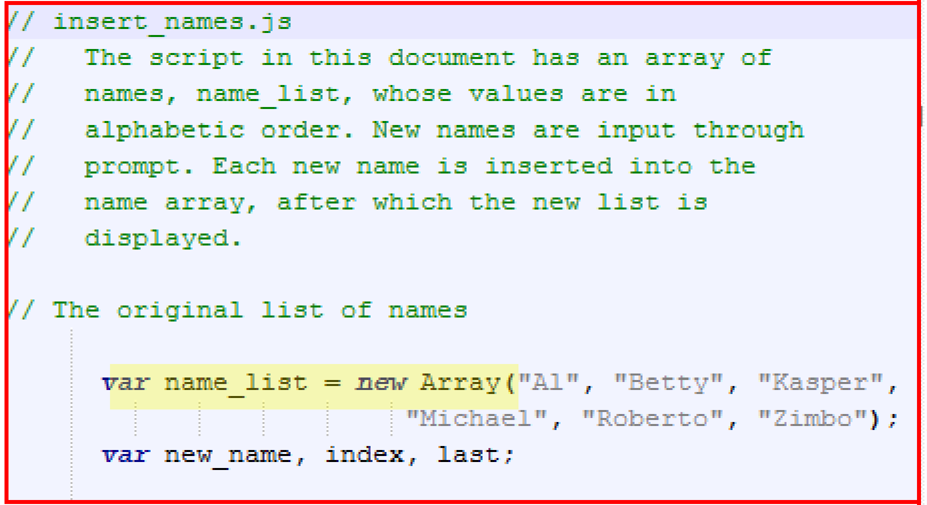
```
myList.length = 150;
```

- Assigning a value to an element that does not exist creates that element.

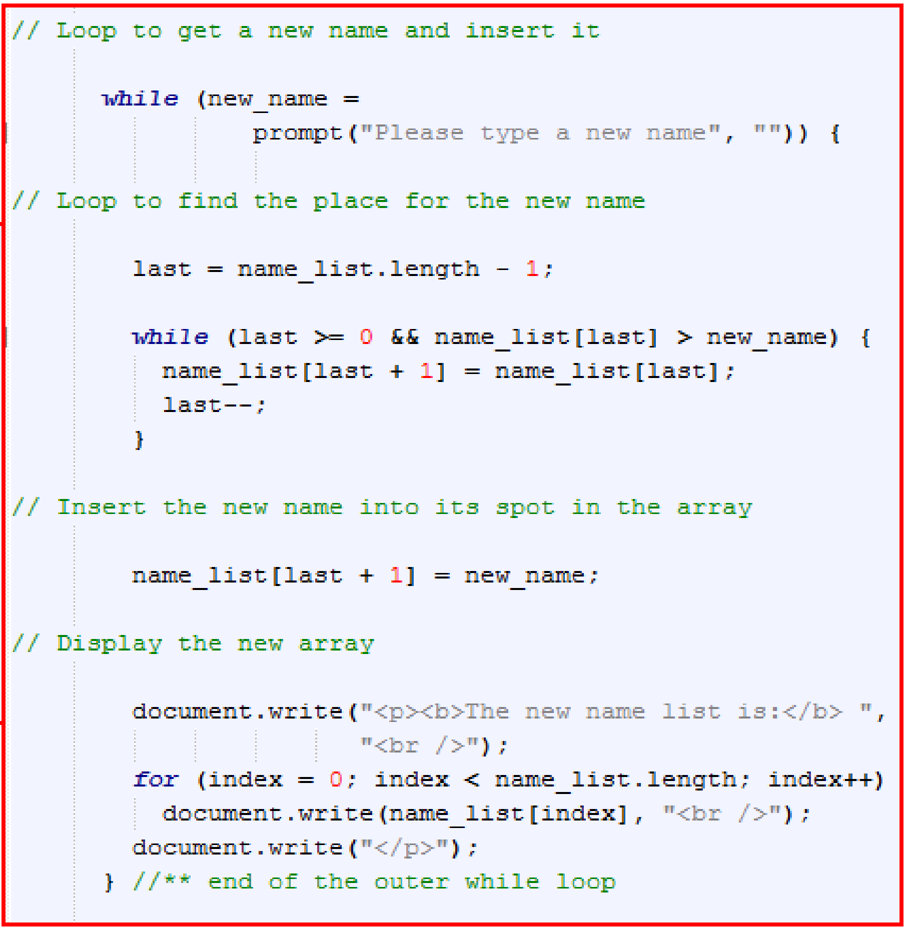
Example 6 – *Array* Object



```
<!DOCTYPE>

<!-- insert_names.html
     A document for insert_names.js
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Name list </title>
  </head>
  <body>
    <script type = "text/javascript"  src = "insert_names.js" >
    </script>
  </body>
</html>
```



```
// insert_names.js
//   The script in this document has an array of
//   names, name_list, whose values are in
//   alphabetic order. New names are input through
//   prompt. Each new name is inserted into the
//   name array, after which the new list is
//   displayed.

// The original list of names

    var name_list = new Array("Al", "Betty", "Kasper",
                      "Michael", "Roberto", "Zimbo");
    var new_name, index, last;
```



```
// Loop to get a new name and insert it

      while (new_name =
                   prompt("Please type a new name", "")) {

// Loop to find the place for the new name

        last = name_list.length - 1;

        while (last >= 0 && name_list[last] > new_name) {
          name_list[last + 1] = name_list[last];
          last--;
        }

// Insert the new name into its spot in the array

        name_list[last + 1] = new_name;

// Display the new array

        document.write("<p><b>The new name list is:</b> ",
                       "<br />");
        for (index = 0; index < name_list.length; index++)
          document.write(name_list[index], "<br />");
        document.write("</p>");
      } //** end of the outer while loop
```

Try it! “Ex6_insert names”

JS Core Object – *Function* Object

- The Function object allows a function to be defined as an object
  - a string is defined at runtime and compiled as a function
  - variables that reference them are treated as other object references
- Format

```
var nameOfFunction = new Function(arguments,
  statements_as_string;)
```

- Example
  - a call to sum(3, 4) will return 7

```
var sum = new Function("a", "b", "return a + b;");
```

- If `sum` is the name of a function,

```
ref_sum = sum;
...
ref_sum();  /* A call to sum */
```

Pattern Matching using RegExp and String Objects

- JavaScript provides two ways to do pattern matching:

  1. Using **`RegExp`** objects
  2. Using methods on **`String`** objects
     - `Search(), match(), replace(), split()`

- *Simple patterns*
  - Two categories of characters in patterns:

    a. normal characters (match themselves)

    b. metacharacters

    ```
    \ | ( ) [ ] { } ^ $ * + ? .
    ```

  - A metacharacter is treated as a normal character if it is backslashed
  - period is a special metacharacter - it matches any character except newline

JS Core Object – *RegExp* Object

- Two ways to create regular expression objects:
  - The literal way

```
var variable_name = /regular expression/options;
```

where options (no spacing after /): `i` (ignore case),
`g` (global search for all occurrences)
`m` (match over multiple lines)

  - The Constructor method

```
var variable_name = new RegExp(
            "regular expression","options");
```

Use escape for literal characters

- Testing the Expression
  - `test()` method
    - Tests for a match in a string and returns either true or false
  - `exec()` method
    - Executes a search for a match in a string and returns an array

Pattern Matching – *String* Object

- User string object methods to test regular expression
- `search`(pattern)
  - Returns the position in the object string of the pattern (position is relative to zero); returns -1 if it fails

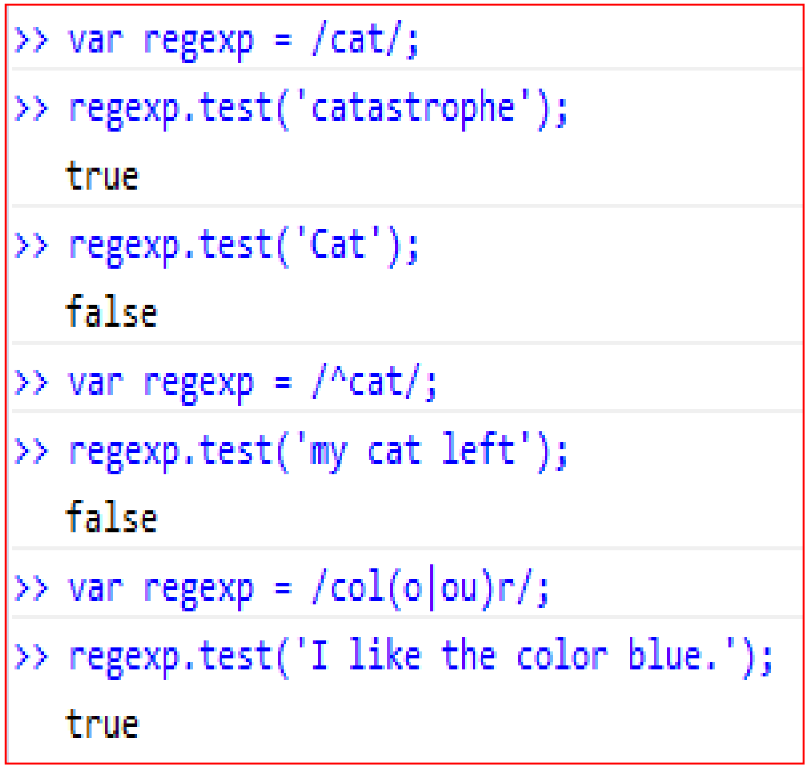
```
var str = "Gluckenheimer";
var position = str.search(/n/);  /* position is now 6 */
```

- `match`(pattern)
  - Returns an array where each element of the array contains each matched pattern that was found.
  - Returns null if no match found

```
var matchArray = new Array();
matchArray = str.match(/luck/);
                  /* matchArray[0] contains "luck" */
```

Metacharacters in Regular Expression

Character	Meaning
\	Escape
^	Indicates the beginning of the string
$	Indicates the end of the string
.	Any single character except newline
|	Alternatives (or)
[	Start of a class
]	End of a class
(	Start of a subpattern
)	End of a subpattern
{	Start of a quantifier
}	End of a quantifier



```
>> var regexp = /cat/;
>> regexp.test('catastrophe');
   true
>> regexp.test('Cat');
   false
>> var regexp = /^cat/;
>> regexp.test('my cat left');
   false
>> var regexp = /col(o|ou)r/;
>> regexp.test('I like the color blue.');
   true
```

Pattern Matching – Character classes

- Character classes
  - Put a sequence of characters in brackets, and it defines a set of characters, any one of which matches
    `[abcd]`
  - Dashes can be used to specify spans of characters in a class
    `[a-z]`
  - A caret at the left end of a class definition means the opposite (not in the set)
    `[^0-9]`
- Character class abbreviations

Abbr.	*Equiv. Pattern*	*Matches*
`\d`	`[0-9]`	a digit
`\D`	`[^0-9]`	not a digit
`\w`	`[A-Za-z_0-9]`	a word character
`\W`	`[^A-Za-z_0-9]`	not a word character
`\s`	`[ \r\t\n\f]`	a whitespace character
`\S`	`[^ \r\t\n\f]`	not a whitespace characte

Pattern Matching - Quantifiers

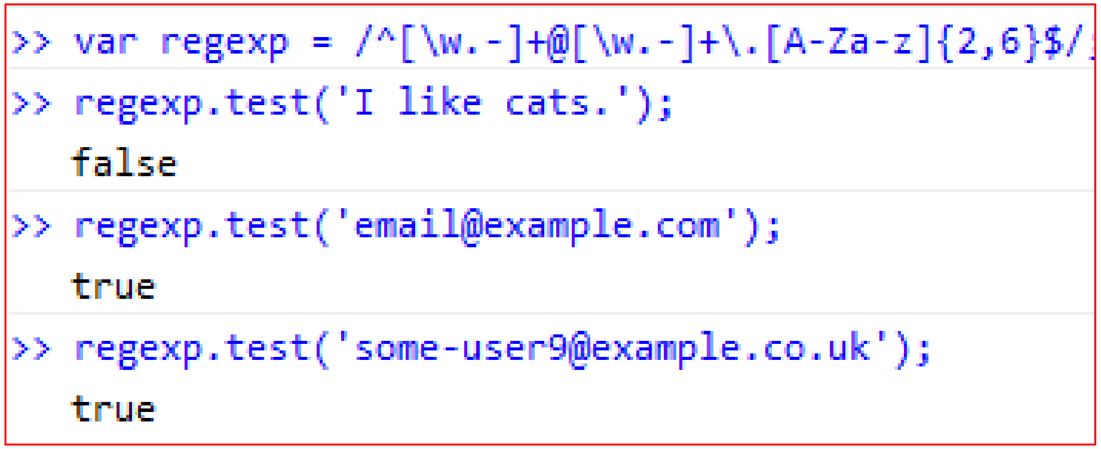
- Quantifiers

Quantifier	*Meaning*
`{n}`	exactly n repetitions
`{m,}`	at least m repetition
`{m, n}`	at least m but not more than n repetitions

- Other quantifiers (for the most commonly used quantifiers)
  - * means zero or more repetitions
    - e.g., `\d*` means zero or more digits
  - + means one or more repetitions
    - e.g., \d+ means one or more digits
  - ? Means zero or one
    - e.g., \d? means zero or one digit

Character Classes - Examples

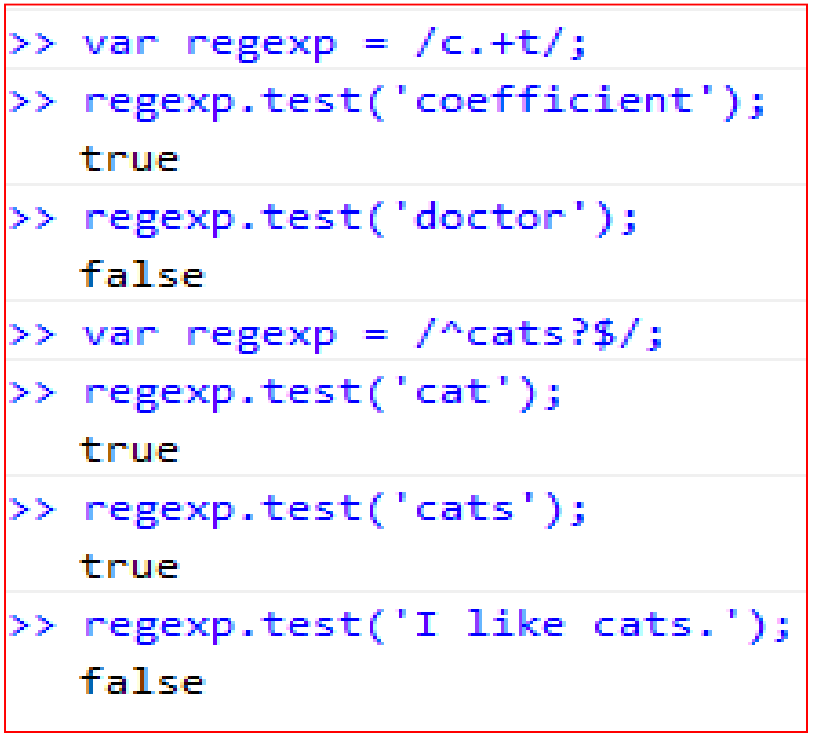
Class	Short cut	Meaning
[0-9]	\d	Any digit
[\f\r\t\n\v]	\s	Any white space
[A-Za-z0-9]	\w	Any word character
[^0-9]	\D	Not a digit
[^\f\r\t\n\v]	\S	Not white space
[^A-Za-z0-9]	\W	Not a word character



```
>> var regexp = /^[\w.-]+@[\w.-]+\.[A-Za-z]{2,6}$/
>> regexp.test('I like cats.');
   false
>> regexp.test('email@example.com');
   true
>> regexp.test('some-user9@example.co.uk');
   true
```

Quantifiers - Examples

Character	Meaning
?	0 or 1
*	0 or more
+	1 or more
{x}	Exactly x occurrences
{x,y}	Between x and y (inclusive)
{x,}	At least x occurrences
x(?=y)	Match x followed by string y
x(?!y)	Match x not followed by string y



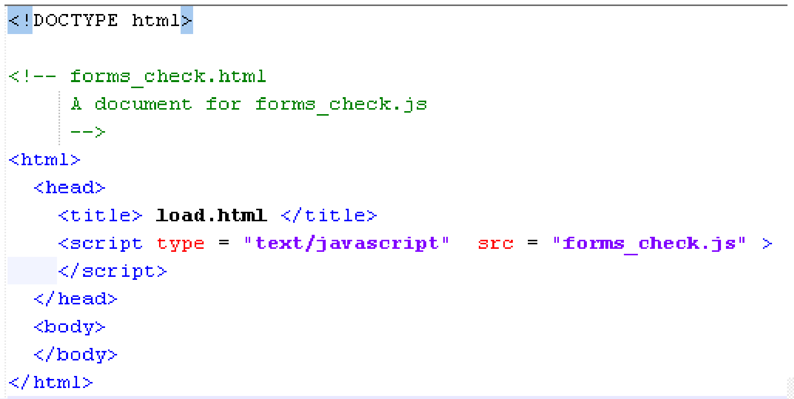
```
>> var regexp = /c.+t/;
>> regexp.test('coefficient');
   true
>> regexp.test('doctor');
   false
>> var regexp = /^cats?$/;
>> regexp.test('cat');
   true
>> regexp.test('cats');
   true
>> regexp.test('I like cats.');
   false
```

Look-ahead match

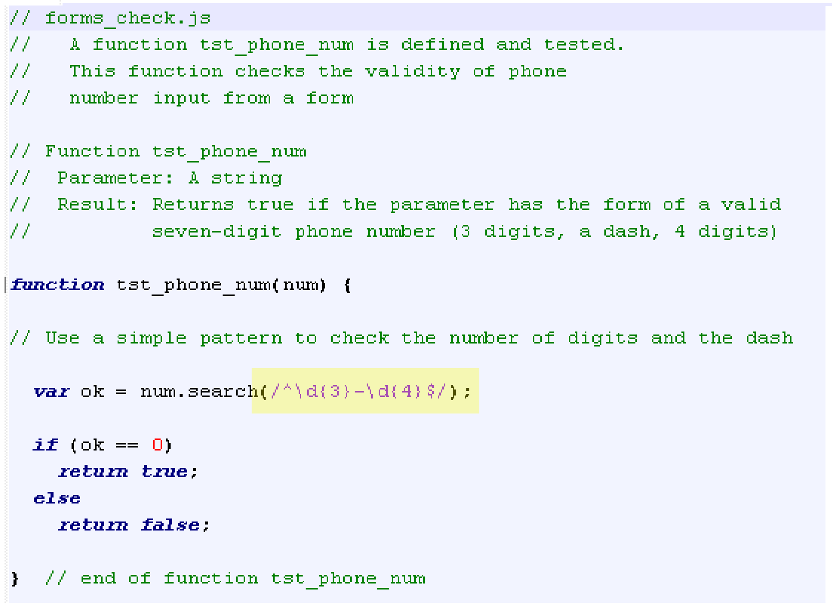
```
>> var regexp1 = /@(?=[\w]).*\.(?=[\w])/;
     regexp1.test ('email@example.co.uk');   True
>> var regexp2 = /(?=.*[\.])(?=.*[@])/;
     regexp2.test ('email@example.co.uk');   True
```

Example 7 – forms check

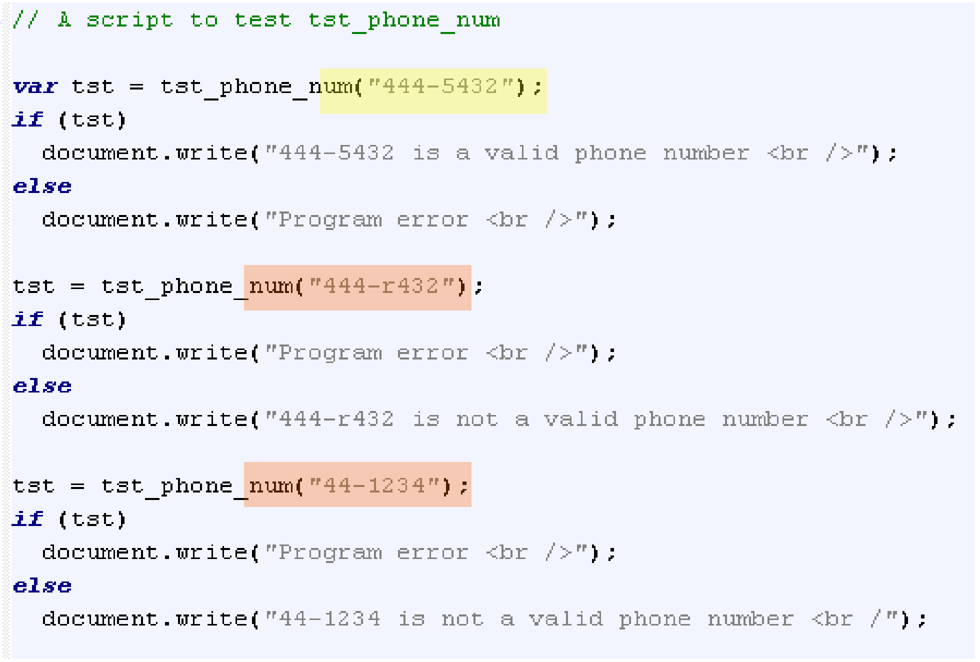
Try it! “Ex7_forms check”



```
<!DOCTYPE html>

<!-- forms_check.html
     A document for forms_check.js
     -->
<html>
  <head>
    <title> load.html </title>
    <script type = "text/javascript"  src = "forms_check.js" >
    </script>
  </head>
  <body>
  </body>
</html>
```



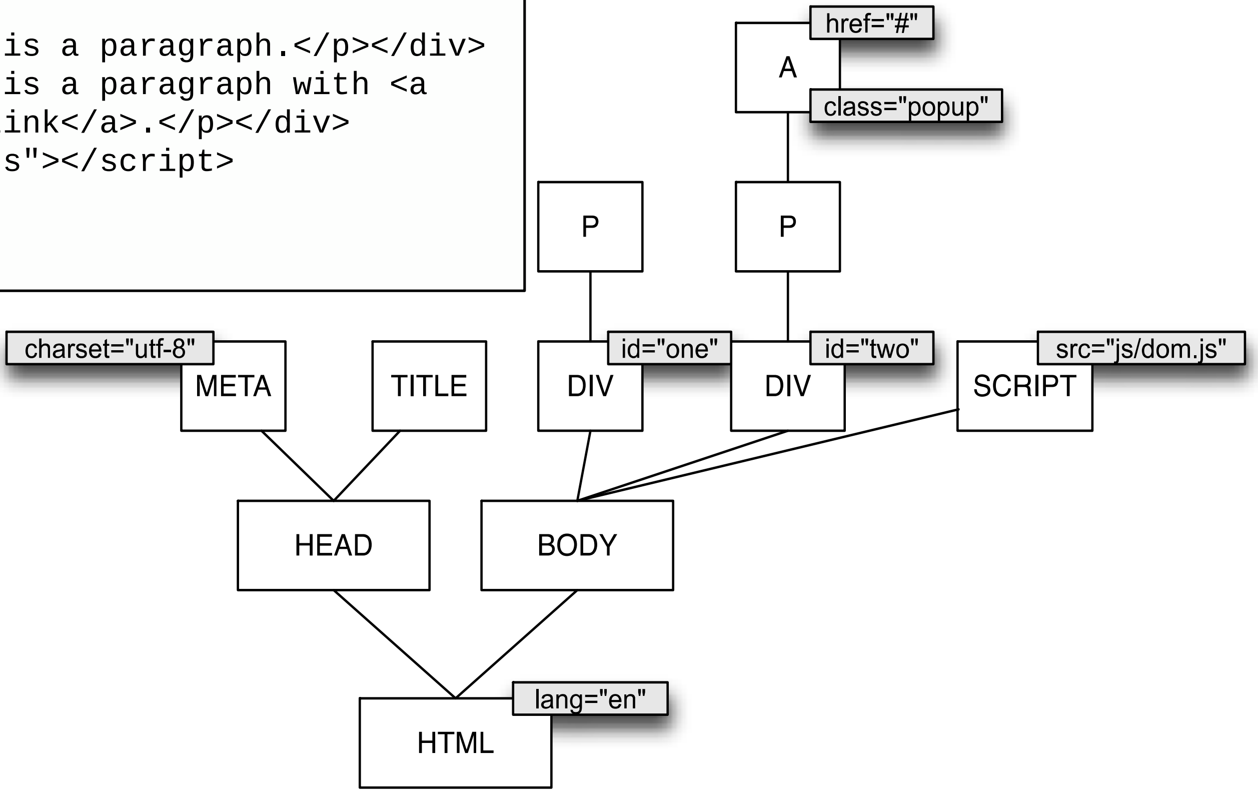
```
// forms_check.js
//   A function tst_phone_num is defined and tested.
//   This function checks the validity of phone
//   number input from a form

// Function tst_phone_num
//  Parameter: A string
//  Result: Returns true if the parameter has the form of a valid
//          seven-digit phone number (3 digits, a dash, 4 digits)

function tst_phone_num(num) {

// Use a simple pattern to check the number of digits and the dash

  var ok = num.search(/^\d{3}-\d{4}$/);

  if (ok == 0)
    return true;
  else
    return false;

}  // end of function tst_phone_num
```



```
// A script to test tst_phone_num

var tst = tst_phone_num("444-5432");
if (tst)
  document.write("444-5432 is a valid phone number <br />");
else
  document.write("Program error <br />");

tst = tst_phone_num("444-r432");
if (tst)
  document.write("Program error <br />");
else
  document.write("444-r432 is not a valid phone number <br />");

tst = tst_phone_num("44-1234");
if (tst)
  document.write("Program error <br />");
else
  document.write("44-1234 is not a valid phone number <br /");
```

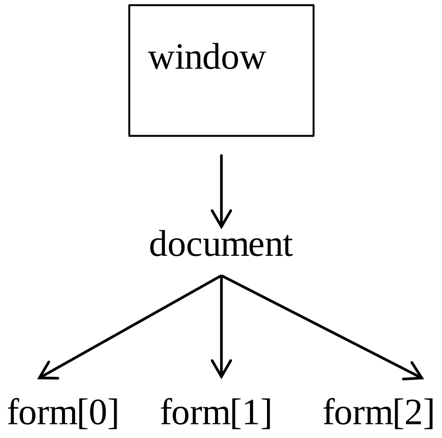
Structure of Document Object Model (DOM)

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>This Is The Title</title>
</head>
<body>
    <div id="one"><p>This is a paragraph.</p></div>
    <div id="two"><p>This is a paragraph with <a
href="#" class="popup">a link</a>.</p></div>
    <script src="js/text.js"></script>
</body>
</html>
```



Accessing Document Objects in DOM

- JavaScript creates an array of all forms, images, links and subordinate objects in a document.
  - Can be accessed by address, name, or method.



Element Access of DOM *form* (1)

1. Access using address reference

   Example (a document with just one form and one widget):

   ```
   <form action = "">
       <input type = "button"  name = "pushMe" />
   </form>
   ```

   – Element address

   ```
   document.forms[0].element[0]
   ```

2. Access using name reference

   Same example with `form` named `myForm`

   ```
   <form name = "myForm"  action = "">
       <input type = "button"  name = "pushMe“ />
   </form>
   ```

   – Element name

   ```
   document.myForm.pushMe
   ```

Element Access of DOM *form* (2)

3. Accessing document objects using DOM methods

- getElementById()
  - returns the element with the specified ID
- getElementsByTagName() (array)
  - returns all elements with a specified tag name
- getElementsByClassName (array)
  - returns all elements with the same class name

Example using `getElementById` method

```
<form action = "">
   <input type = "button"  id = "pushMe" />
</form>
```

- DOM object Id referenced

```
document.getElementById("pushMe")
```

Element Access of other DOM Objects

- Checkboxes and radio button have an implicit array, which has their name

```
<form id = "topGroup">
  <input type="checkbox“ name="toppings“  value="olives" />
          ...
  <input type="checkbox" name="toppings“  value="tomatoes"/>
</form>
```

```
var numChecked = 0;

var dom = document.getElementById("topGroup");

for (index = 0; index < dom.toppings.length; index++)
   if (dom.toppings[index].checked)
       numChecked++;
```

Events and Event Handling

Event	*Tag Attribute*
`blur`	`onblur`
`change`	`onchange`
`click`	`onclick`
`dblclick`	`ondblclick`
`focus`	`onfocus`
`keydown`	`onkeydown`
`keypress`	`onkeypress`
`keyup`	`onkeyup`
`load`	`onload`
`mousedown`	`onmousedown`
`mousemove`	`onmousemove`
`mouseout`	`onmouseout`
`mouseover`	`onmouseover`
`mouseup`	`onmouseup`
`reset`	`onreset`
`select`	`onselect`
`submit`	`onsubmit`
`unload`	`onunload`

- An event is a notification that something specific has occurred, either with the browser or an action of the browser user
- An event handler is a script that is implicitly executed in response to the occurrence of an event
- The process of connecting an event handler to an event is called registration
- Don’t use document.write in an event handler, because the output may go on top of the display

Event Handling of *click* event

- The same attribute can appear in several different tags

  e.g., The `onclick` attribute can be in `<a>` and `<input>`
- A text element gets focus :
  - When the user puts the mouse cursor over it and presses the left button
  - When the user tabs to the elements
- Event handlers can be registered by assigning the event handler script to an event tag attribute

```
onclick = "alert('Mouse click!');"

onclick = "myHandler();"
```

- Inline Event Listeners
  - Simple but intrusive to HTML

```
<form action=“ “ method="post" onsubmit="validateForm();">
<a href="somepage.html" onclick="doSomething();">Some Link</a>
```

Handling Events from Button Elements

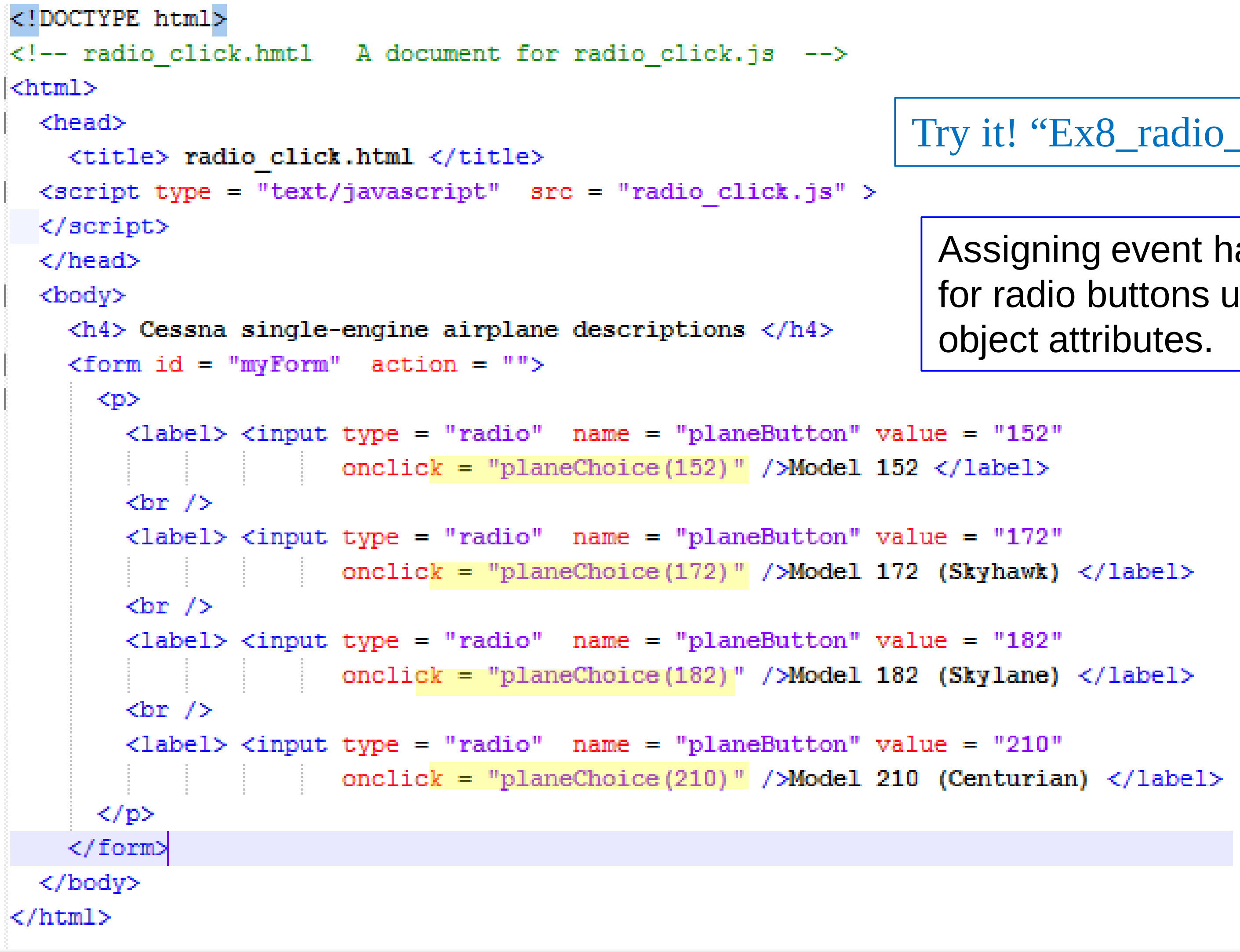
- Plain Buttons
  - use the `onclick` property
- Radio buttons
  - If the handler is registered in the markup, the particular button that was clicked can be sent to the handler as an event attribute

    e.g., if `planeChoice` is the name of the handler and the value of a button is `172`, use

    ```
    onclick = ″planeChoice(172)″
    ```
- One way to register an event handler for a radio button
  - Assign the address of the handler function to the event property of the object associated with the HTML element.

    ```
    var dom = document.getElementById(″myForm″)
    dom.elements[0].onclick = planeChoice;
    ```

Example 8 – Event handling by object attributes

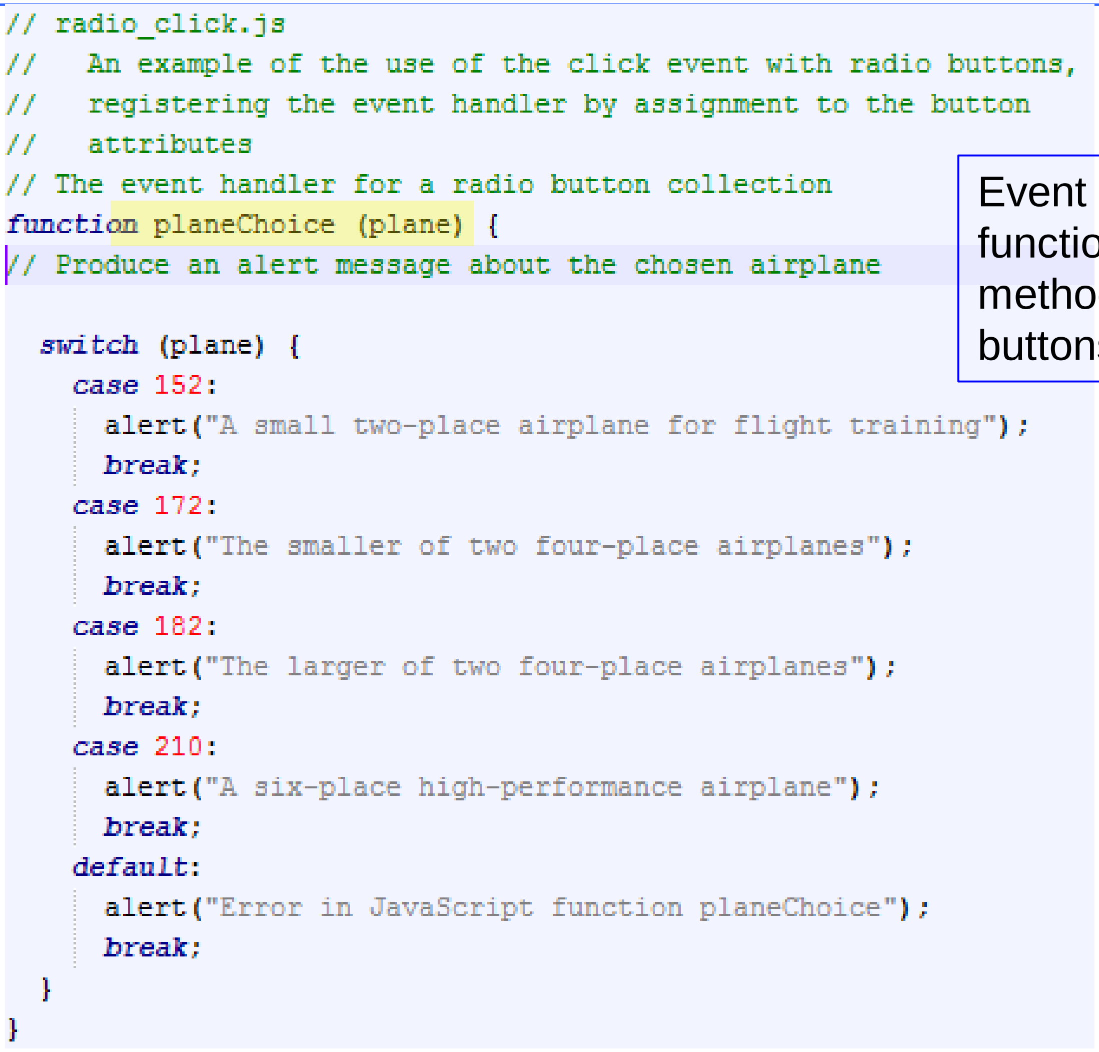


```
<!DOCTYPE html>
<!-- radio_click.hmtl   A document for radio_click.js  -->
<html>
  <head>
    <title> radio_click.html </title>
  <script type = "text/javascript"  src = "radio_click.js" >
  </script>
  </head>
  <body>
    <h4> Cessna single-engine airplane descriptions </h4>
    <form id = "myForm"  action = "">
      <p>
        <label> <input type = "radio"  name = "planeButton" value = "152"
                onclick = "planeChoice(152)" />Model 152 </label>
        <br />
        <label> <input type = "radio"  name = "planeButton" value = "172"
                onclick = "planeChoice(172)" />Model 172 (Skyhawk) </label>
        <br />
        <label> <input type = "radio"  name = "planeButton" value = "182"
                onclick = "planeChoice(182)" />Model 182 (Skylane) </label>
        <br />
        <label> <input type = "radio"  name = "planeButton" value = "210"
                onclick = "planeChoice(210)" />Model 210 (Centurian) </label>
      </p>
    </form>
  </body>
</html>
```

Try it! “Ex8_radio_click”

Assigning event handler for radio buttons using object attributes.

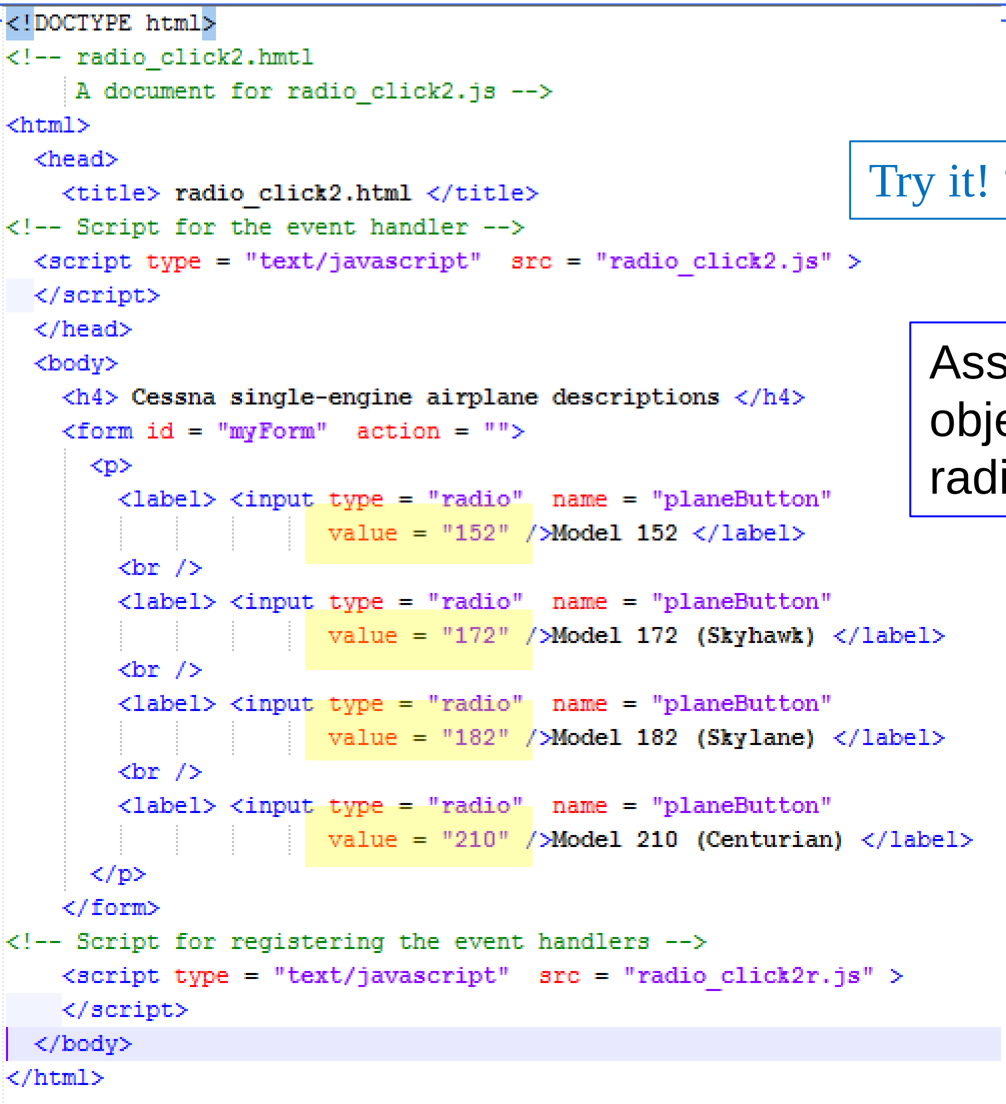
Example 8 – Event handler for *radio buttons*



```
// radio_click.js
//   An example of the use of the click event with radio buttons,
//   registering the event handler by assignment to the button
//   attributes
// The event handler for a radio button collection
function planeChoice (plane) {
// Produce an alert message about the chosen airplane

  switch (plane) {
    case 152:
      alert("A small two-place airplane for flight training");
      break;
    case 172:
      alert("The smaller of two four-place airplanes");
      break;
    case 182:
      alert("The larger of two four-place airplanes");
      break;
    case 210:
      alert("A six-place high-performance airplane");
      break;
    default:
      alert("Error in JavaScript function planeChoice");
      break;
  }
}
```

Event handler using function as event method for the radio buttons

Example 9 – Event handing by object property

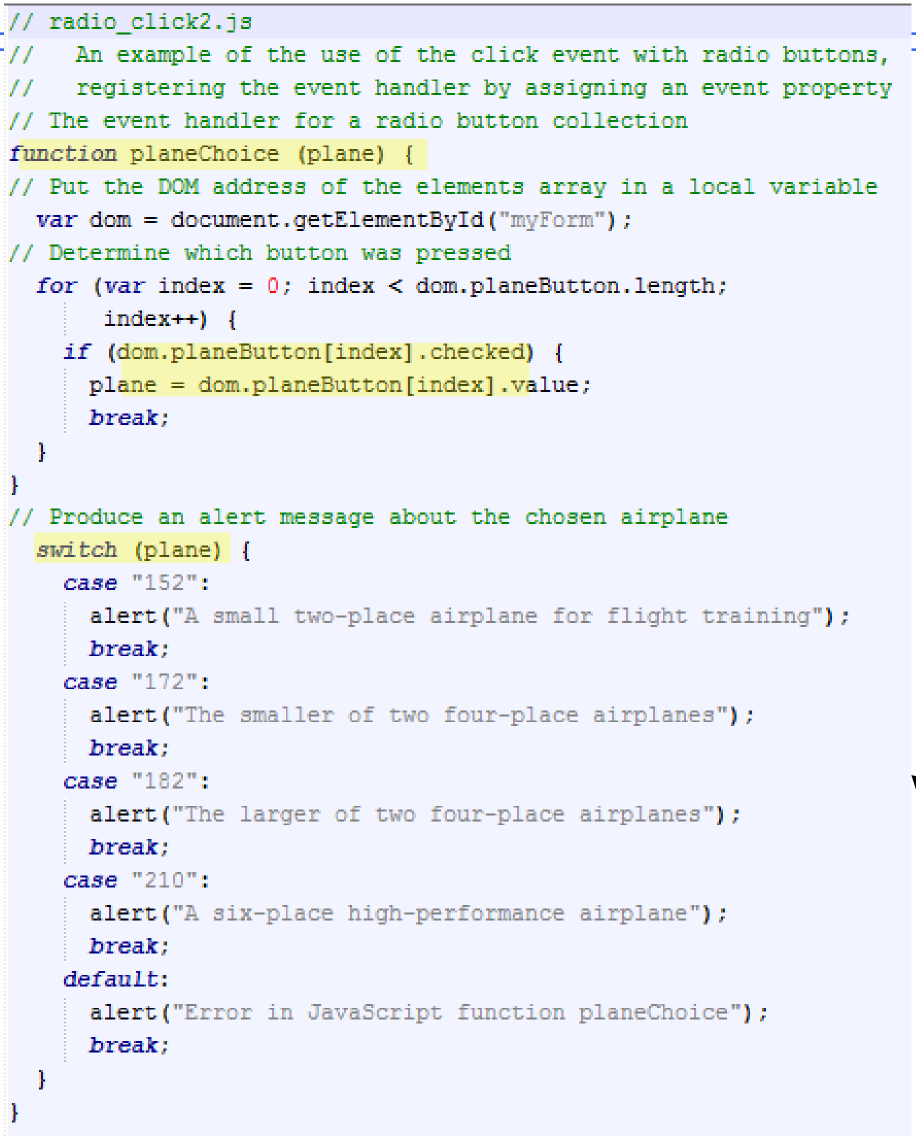


```
<!DOCTYPE html>
<!-- radio_click2.hmtl
     A document for radio_click2.js -->
<html>
  <head>
    <title> radio_click2.html </title>
<!-- Script for the event handler -->
  <script type = "text/javascript"  src = "radio_click2.js" >
  </script>
  </head>
  <body>
    <h4> Cessna single-engine airplane descriptions </h4>
    <form id = "myForm"  action = "">
      <p>
        <label> <input type = "radio"  name = "planeButton"
                       value = "152" />Model 152 </label>
        <br />
        <label> <input type = "radio"  name = "planeButton"
                       value = "172" />Model 172 (Skyhawk) </label>
        <br />
        <label> <input type = "radio"  name = "planeButton"
                       value = "182" />Model 182 (Skylane) </label>
        <br />
        <label> <input type = "radio"  name = "planeButton"
                       value = "210" />Model 210 (Centurian) </label>
      </p>
    </form>
<!-- Script for registering the event handlers -->
    <script type = "text/javascript"  src = "radio_click2r.js" >
    </script>
  </body>
</html>
```

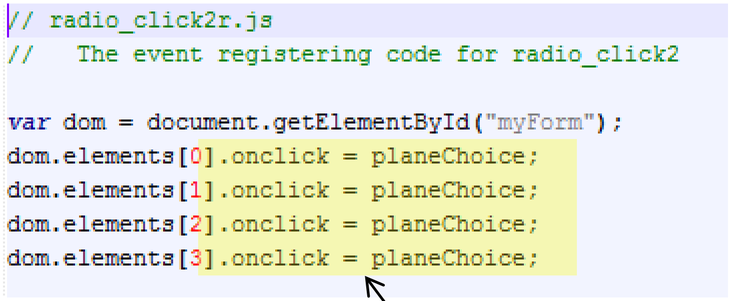
Try it! “Ex9_radio_click2”

Assigning values to object property of radio buttons

Example 9 – Event registration and handler for *radio_clicks*



```
// radio_click2.js
//   An example of the use of the click event with radio buttons,
//   registering the event handler by assigning an event property
// The event handler for a radio button collection
function planeChoice (plane) {
// Put the DOM address of the elements array in a local variable
  var dom = document.getElementById("myForm");
// Determine which button was pressed
  for (var index = 0; index < dom.planeButton.length;
        index++) {
    if (dom.planeButton[index].checked) {
      plane = dom.planeButton[index].value;
      break;
  }
}
// Produce an alert message about the chosen airplane
  switch (plane) {
    case "152":
      alert("A small two-place airplane for flight training");
      break;
    case "172":
      alert("The smaller of two four-place airplanes");
      break;
    case "182":
      alert("The larger of two four-place airplanes");
      break;
    case "210":
      alert("A six-place high-performance airplane");
      break;
    default:
      alert("Error in JavaScript function planeChoice");
      break;
  }
}
```



```
// radio_click2r.js
//   The event registering code for radio_click2

var dom = document.getElementById("myForm");
dom.elements[0].onclick = planeChoice;
dom.elements[1].onclick = planeChoice;
dom.elements[2].onclick = planeChoice;
dom.elements[3].onclick = planeChoice;
```

Registration of event handler to event property for radio_clicks

Handling of events using event attributes of radio_clicks

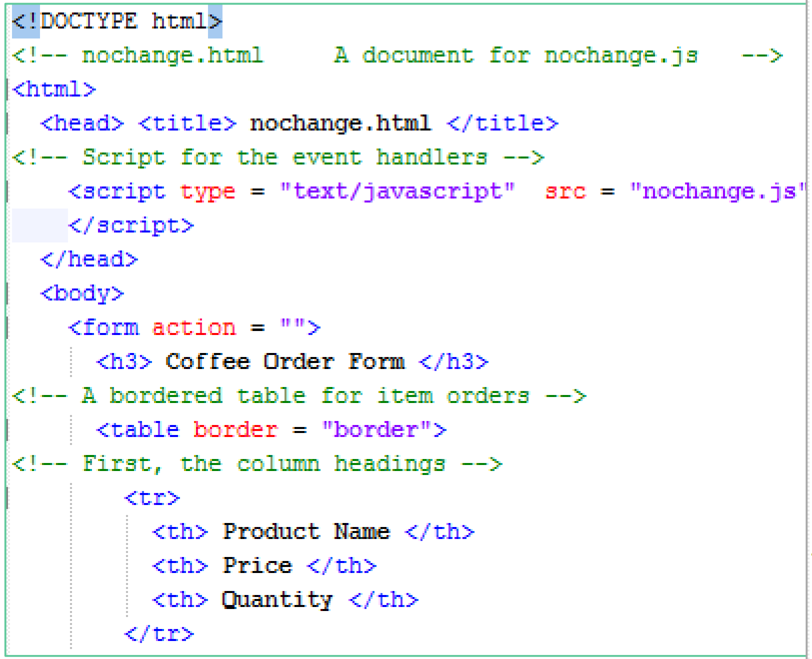
Handling Events in Textbox and Form Elements

- The Focus Event
  - Can be used to detect illicit changes to a text box by blurring the element every time the element acquires focus

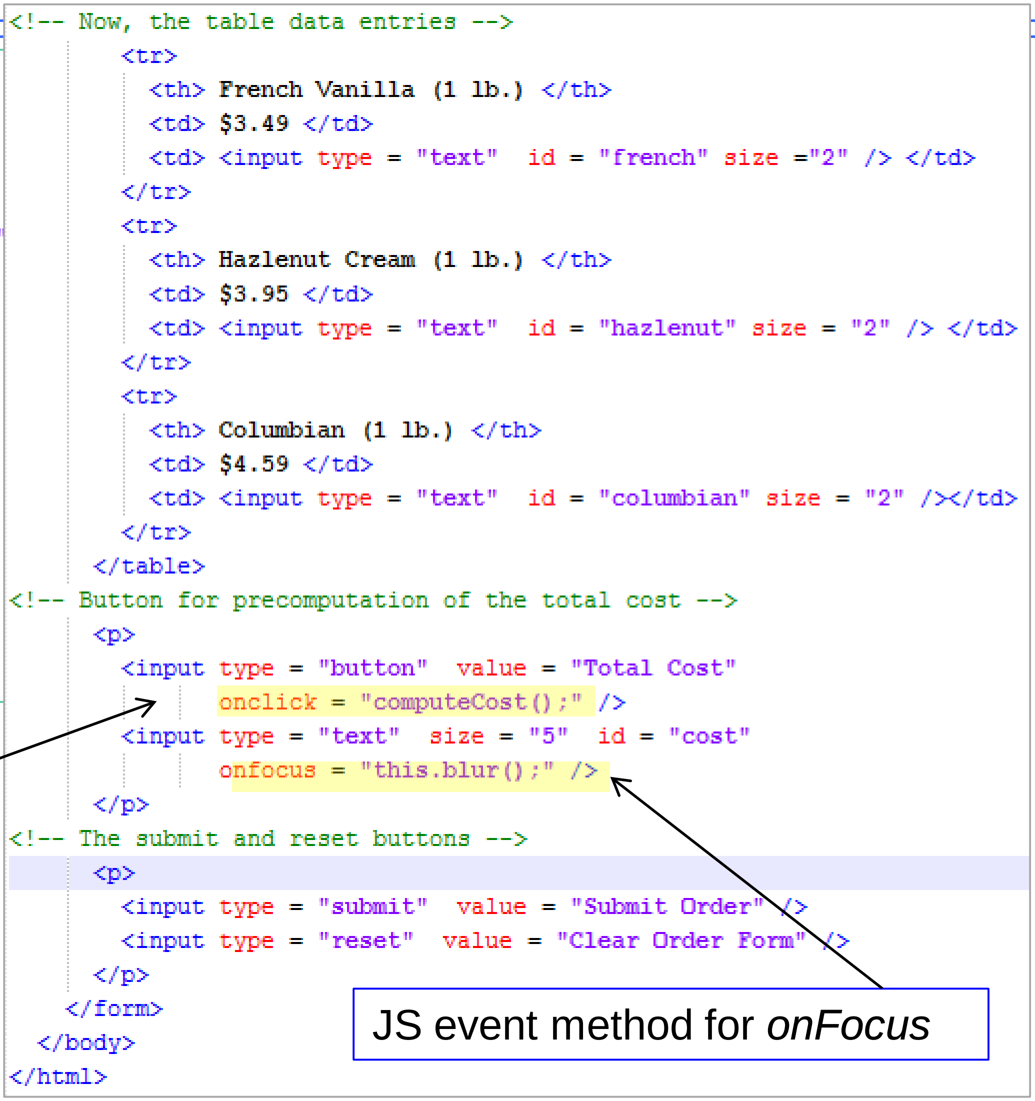
    ```
    <input type="text" onfocus="myFunction()">
    ```

  - The blur() method is used to remove focus from an element.
- Checking Form inputs using the *select()* method
  - A good use of JavaScript, to check for errors in form input before it is sent to the server for processing
  - This saves both: (1) Server time, and (2) Internet time
  - Things that must be done:
    1. Detect the error and produce an `alert` message
    2. Select the element (the `select` function)
  - The `select` function highlights the text in the element
  - To keep the form active after the event handler is finished, the handler must return `false`

Example 10 – Event handling in Textbox



```html
<!DOCTYPE html>
<!-- nochange.html     A document for nochange.js   -->
<html>
  <head> <title> nochange.html </title>
<!-- Script for the event handlers -->
    <script type = "text/javascript"  src = "nochange.js"
    </script>
  </head>
  <body>
    <form action = "">
      <h3> Coffee Order Form </h3>
<!-- A bordered table for item orders -->
      <table border = "border">
<!-- First, the column headings -->
        <tr>
          <th> Product Name </th>
          <th> Price </th>
          <th> Quantity </th>
        </tr>
```



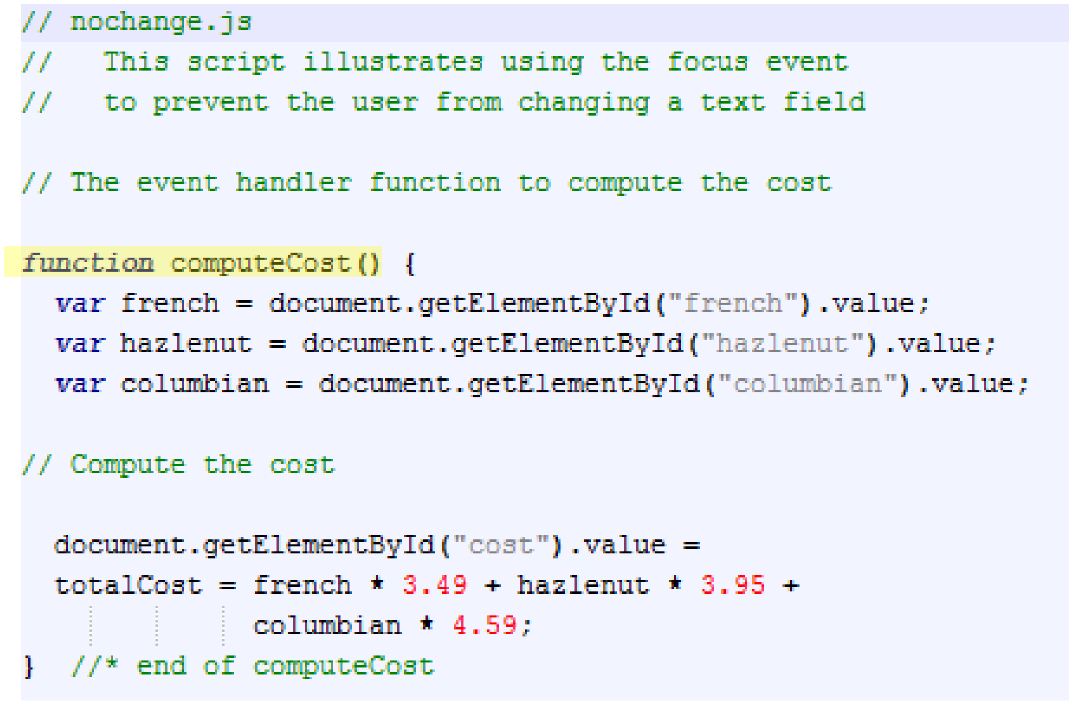
```html
<!-- Now, the table data entries -->
        <tr>
          <th> French Vanilla (1 lb.) </th>
          <td> $3.49 </td>
          <td> <input type = "text"  id = "french" size ="2" /> </td>
        </tr>
        <tr>
          <th> Hazlenut Cream (1 lb.) </th>
          <td> $3.95 </td>
          <td> <input type = "text"  id = "hazlenut" size = "2" /> </td>
        </tr>
        <tr>
          <th> Columbian (1 lb.) </th>
          <td> $4.59 </td>
          <td> <input type = "text"  id = "columbian" size = "2" /></td>
        </tr>
      </table>
<!-- Button for precomputation of the total cost -->
      <p>
        <input type = "button"  value = "Total Cost"
               onclick = "computeCost();" />
        <input type = "text"  size = "5"  id = "cost"
               onfocus = "this.blur();" />
      </p>
<!-- The submit and reset buttons -->
      <p>
        <input type = "submit"  value = "Submit Order" />
        <input type = "reset"  value = "Clear Order Form" />
      </p>
    </form>
  </body>
</html>
```

Assigning event method to *onClick*

Try it! "Ex10_nochange"

JS event method for *onFocus*

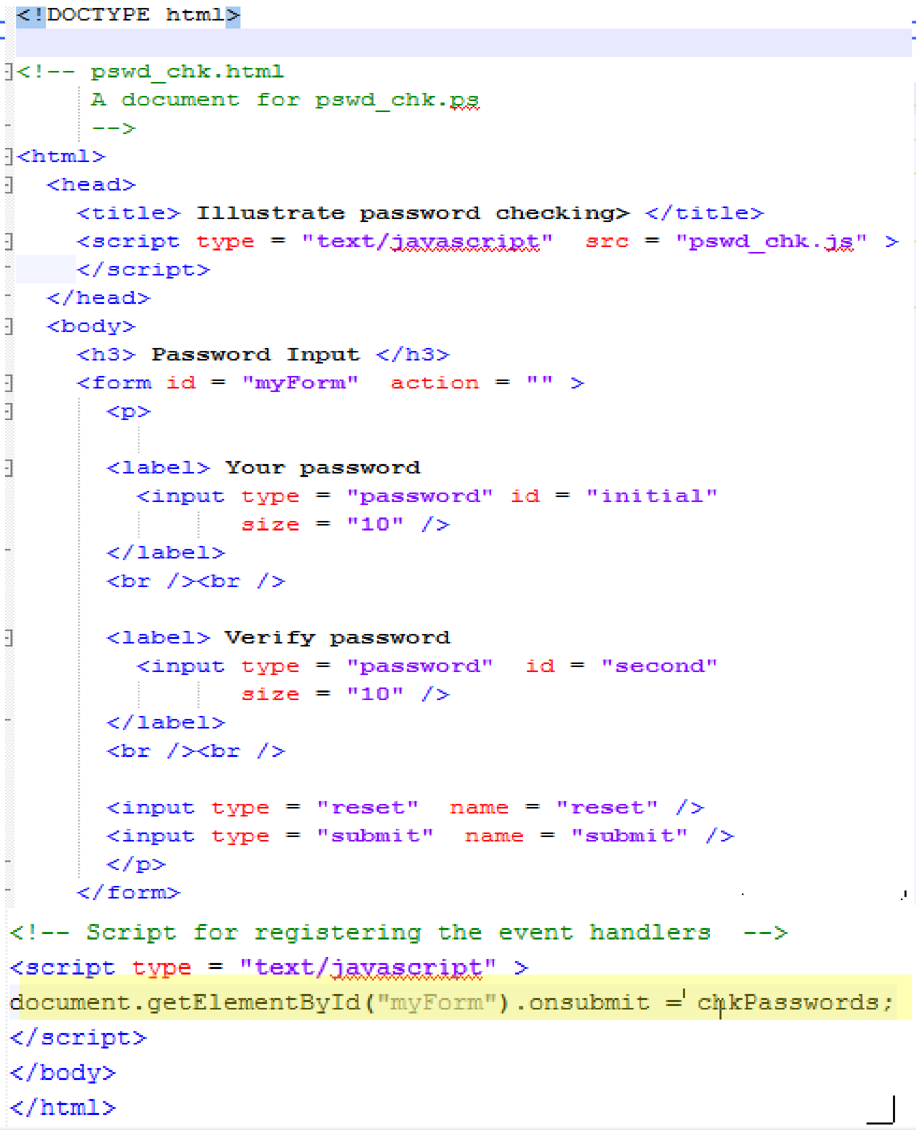
Example 10 – Event handler to compute “cost”



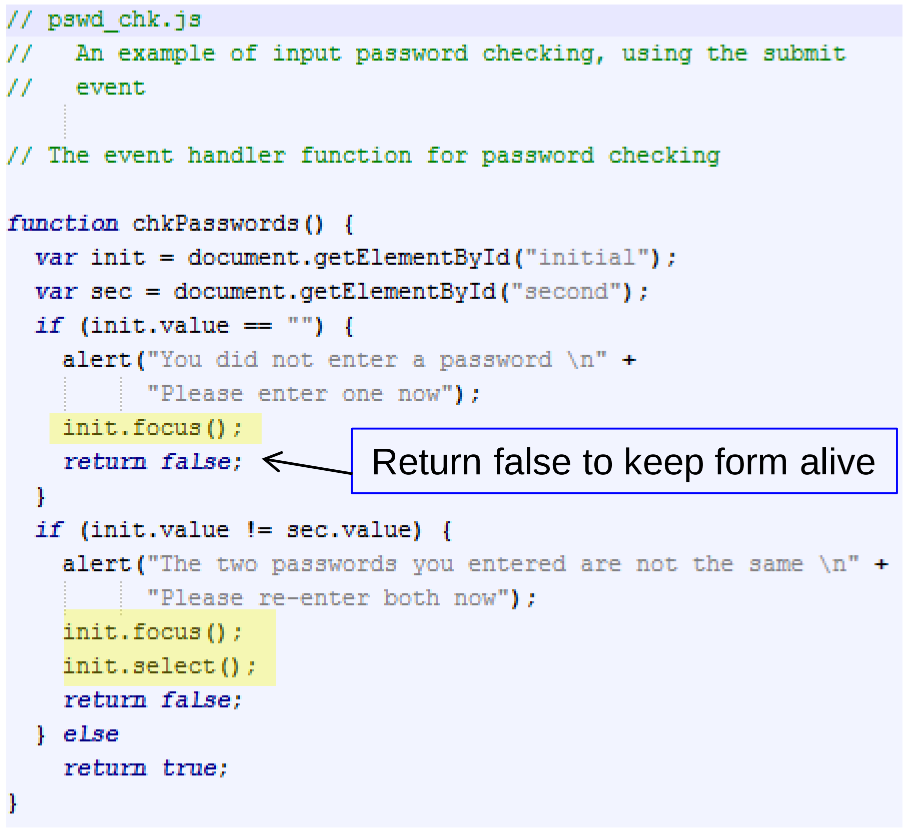
```
// nochange.js
//   This script illustrates using the focus event
//   to prevent the user from changing a text field

// The event handler function to compute the cost

function computeCost() {
  var french = document.getElementById("french").value;
  var hazlenut = document.getElementById("hazlenut").value;
  var columbian = document.getElementById("columbian").value;

// Compute the cost

  document.getElementById("cost").value =
  totalCost = french * 3.49 + hazlenut * 3.95 +
            columbian * 4.59;
}  //* end of computeCost
```

Event handler using function as event method

Example 11- Event handling in Forms



```html
<!DOCTYPE html>

<!-- pswd_chk.html
     A document for pswd_chk.ps
     -->
<html>
  <head>
    <title> Illustrate password checking> </title>
    <script type = "text/javascript"  src = "pswd_chk.js" >
    </script>
  </head>
  <body>
    <h3> Password Input </h3>
    <form id = "myForm"  action = "" >
      <p>

      <label> Your password
        <input type = "password" id = "initial"
               size = "10" />
      </label>
      <br /><br />

      <label> Verify password
        <input type = "password"  id = "second"
               size = "10" />
      </label>
      <br /><br />

      <input type = "reset"  name = "reset" />
      <input type = "submit"  name = "submit" />
      </p>
    </form>
<!-- Script for registering the event handlers  -->
<script type = "text/javascript" >
document.getElementById("myForm").onsubmit = chkPasswords;
</script>
</body>
</html>
```

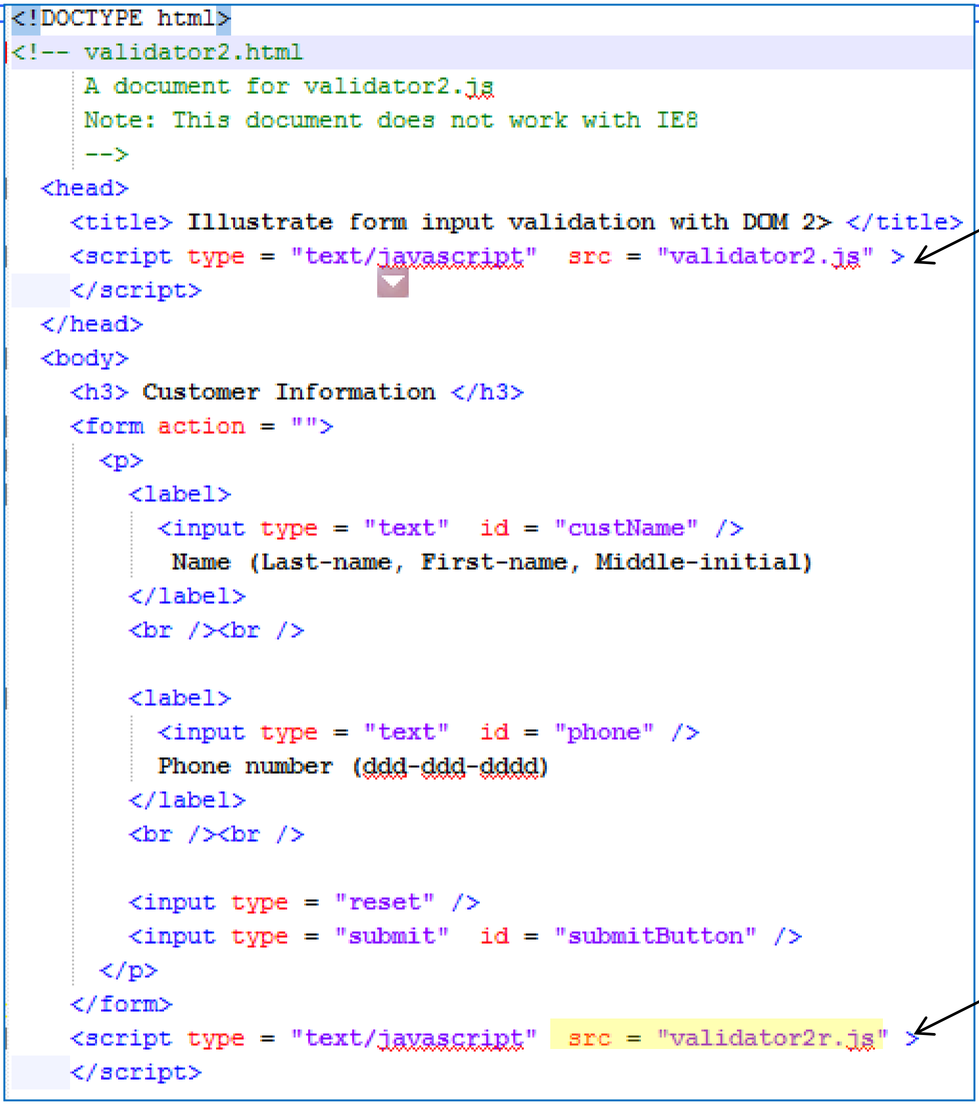


```javascript
// pswd_chk.js
//   An example of input password checking, using the submit
//   event

// The event handler function for password checking

function chkPasswords() {
  var init = document.getElementById("initial");
  var sec = document.getElementById("second");
  if (init.value == "") {
    alert("You did not enter a password \n" +
          "Please enter one now");
    init.focus();
    return false;
  }
  if (init.value != sec.value) {
    alert("The two passwords you entered are not the same \n" +
          "Please re-enter both now");
    init.focus();
    init.select();
    return false;
  } else
    return true;
}
```

Return false to keep form alive

Try it! “Ex11_pswd_chk”

Example 12- Adding event listeners in DOM



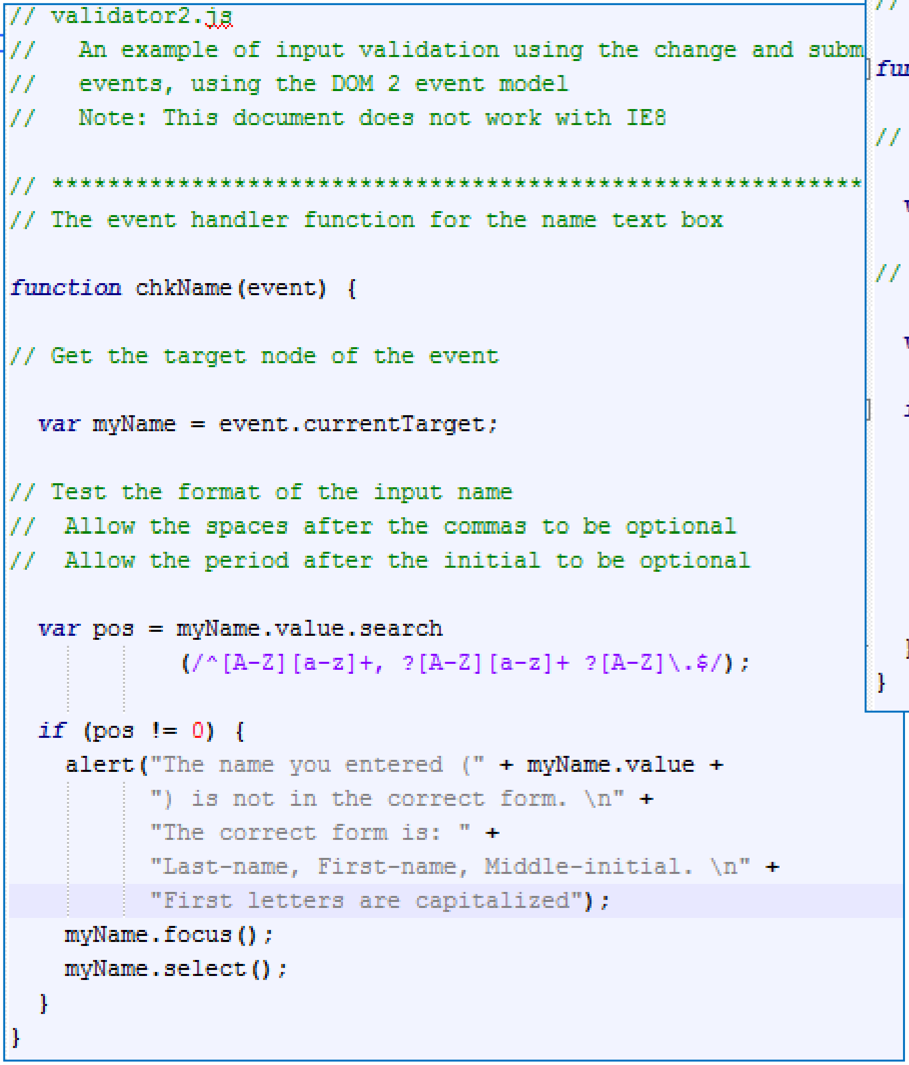
```
<!DOCTYPE html>
<!-- validator2.html
     A document for validator2.js
     Note: This document does not work with IE8
     -->
  <head>
    <title> Illustrate form input validation with DOM 2> </title>
    <script type = "text/javascript"  src = "validator2.js" >
    </script>
  </head>
  <body>
    <h3> Customer Information </h3>
    <form action = "">
      <p>
        <label>
          <input type = "text"  id = "custName" />
           Name (Last-name, First-name, Middle-initial)
        </label>
        <br /><br />

        <label>
          <input type = "text"  id = "phone" />
          Phone number (ddd-ddd-dddd)
        </label>
        <br /><br />

        <input type = "reset" />
        <input type = "submit"  id = "submitButton" />
      </p>
    </form>
    <script type = "text/javascript"  src = "validator2r.js" >
    </script>
```

External functions loaded first

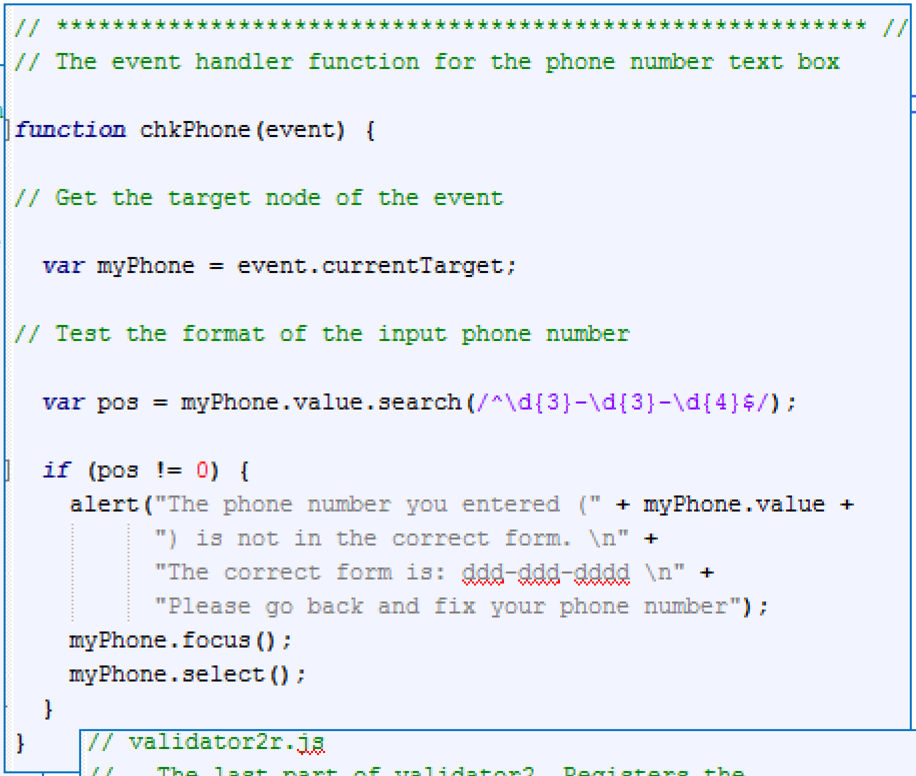
Try it! "Ex12_validator2"

Event registration loaded last

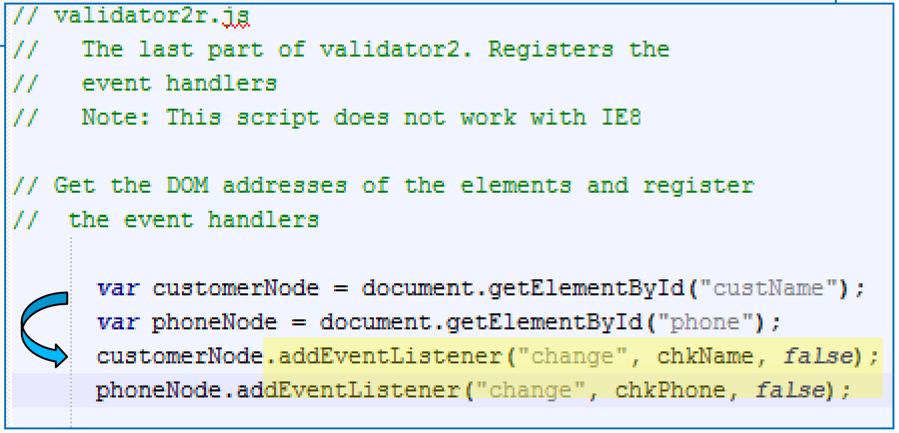
Example 12



```
// validator2.js
//   An example of input validation using the change and subm
//   events, using the DOM 2 event model
//   Note: This document does not work with IE8

// ***********************************************************
// The event handler function for the name text box

function chkName(event) {

// Get the target node of the event

  var myName = event.currentTarget;

// Test the format of the input name
//  Allow the spaces after the commas to be optional
//  Allow the period after the initial to be optional

  var pos = myName.value.search
          (/^[A-Z][a-z]+, ?[A-Z][a-z]+ ?[A-Z]\.$/);

  if (pos != 0) {
    alert("The name you entered (" + myName.value +
          ") is not in the correct form. \n" +
          "The correct form is: " +
          "Last-name, First-name, Middle-initial. \n" +
          "First letters are capitalized");
    myName.focus();
    myName.select();
  }
}
```



```
// ************************************************************ //
// The event handler function for the phone number text box

function chkPhone(event) {

// Get the target node of the event

  var myPhone = event.currentTarget;

// Test the format of the input phone number

  var pos = myPhone.value.search(/^\d{3}-\d{3}-\d{4}$/);

  if (pos != 0) {
    alert("The phone number you entered (" + myPhone.value +
          ") is not in the correct form. \n" +
          "The correct form is: ddd-ddd-dddd \n" +
          "Please go back and fix your phone number");
    myPhone.focus();
    myPhone.select();
  }
}
```



```
// validator2r.js
//   The last part of validator2. Registers the
//   event handlers
//   Note: This script does not work with IE8

// Get the DOM addresses of the elements and register
//  the event handlers

  var customerNode = document.getElementById("custName");
  var phoneNode = document.getElementById("phone");
  customerNode.addEventListener("change", chkName, false);
  phoneNode.addEventListener("change", chkPhone, false);
```